



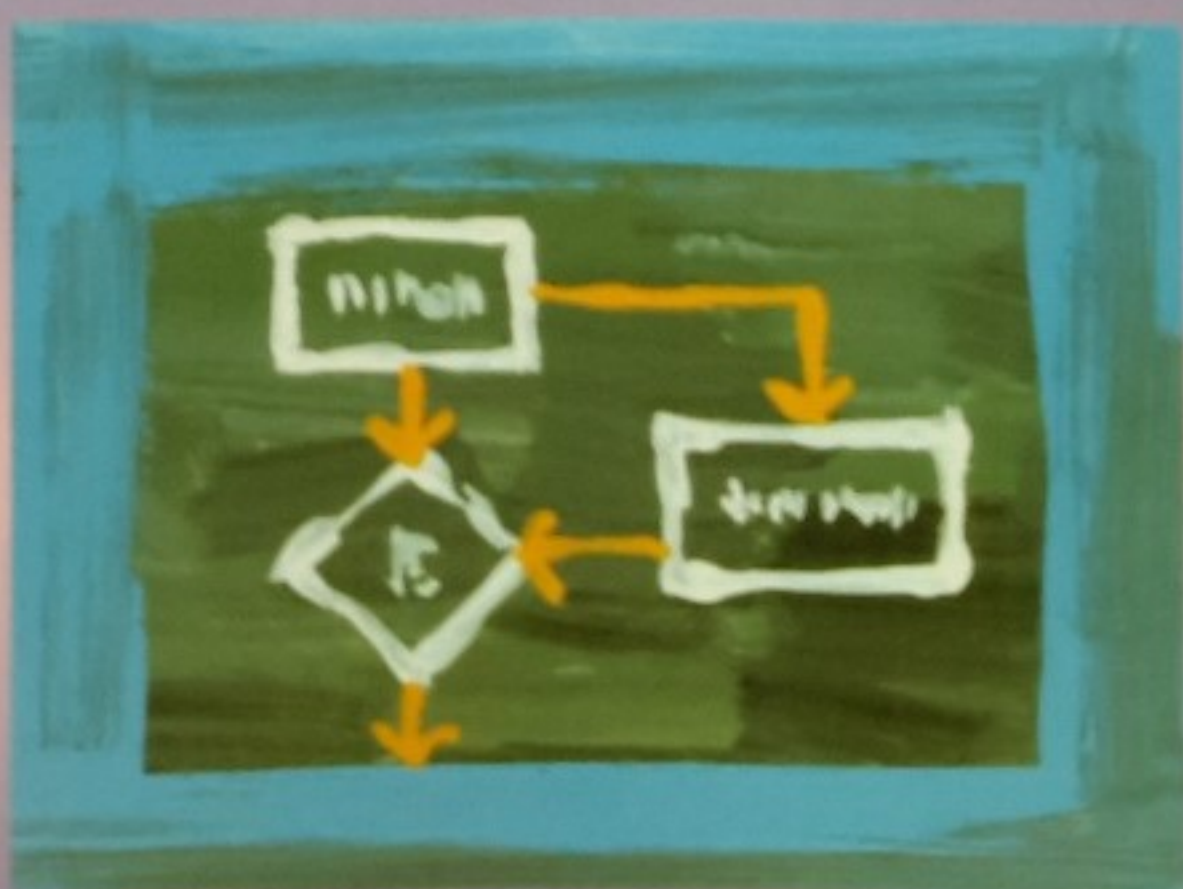
GRUPUL EDITORIAL
CORINT

ELEMENTE DE PROGRAMARE

LECTII PRACTICE PENTRU ELEVII DE GIMNAZIU

Mioara GHEORGHE

Mariana KISCK



AUXILIARE ȘCOLARE

ELEMENTE DE PROGRAMARE lecții practice pentru elevii de gimnaziu

Prin aceste **lecții practice**, copiii sunt orientați, pas cu pas, să gândească rezolvarea problemelor cu calculatorul, de la analiza datelor și până la compunerea testelor care probează calitatea unui program. Lucrarea cuprinde atât probleme clasice, cât și probleme sau situații noi, adaptate vârstei. În alegerea și prezentarea acestora s-a urmărit și calitatea „produsului” realizat de programator: designul, ușurința în utilizare, performanța sau eficiența în timp a programului.

Conținutul lucrării este **accesibil** elevilor de gimnaziu, atât prin nivelul cunoștințelor prezentate, cât și prin stilul abordat, bazat pe comunicare directă, făcând apel la intuiție și creativitate.

Fiecare temă are un număr de idei, sfaturi practice sau trucuri utile oricărui programator. Pentru verificarea abilităților de gândire sau de programare dobândite, lucrarea conține algoritmi propuși spre descifrare sau programe care așteaptă să fie depanate (toate scrise în limbajul Pascal — cel mai folosit la nivel gimnazial).

ISBN: 973-653-527-4



120.000

Mioara GHEORGHE
Mariana KISCK

ELEMENTE DE PROGRAMARE

Lecții practice pentru elevii de gimnaziu



GRUPUL EDITORIAL
CORINT

București, 2003

Redactor:

Alice Raluca Petrescu

Tehnoredactare:

Corina Roncea, Jora Grecea

Coperta:

Walter Riess

Date despre autori:

MIOARA GHEORGHE – profesor gradul I la Colegiul Național „Mihai Viteazul” București, profesor metodist la ISMB, autor de manuale de informatică și auxiliare pentru clasele liceale, colaborator la reviste de specialitate, membru în diverse comisii la olimpiade și concursuri naționale de informatică. A elaborat partea I a acestei lucrări.

MARIANA KISCH – inginer, profesor de informatică la Cygnus Computer București, autor a unor cărți de programare în Turbo Pascal. A elaborat partea a II-a a acestei lucrări.

Descrierea CIP a Bibliotecii Naționale a României
GHEORGHE, MIOARA

Elemente de programare: lecții practice pentru elevii de gimnaziu / Mioara Gheorghe, Mariana Kisch. - București: Corint, 2003

ISBN 973-653-527-4

I. Kisch, Mariana

004.42

Pentru comenzi și informații adresați-vă la:

Editura CORINT

Difuzare:

Str. Teodosie Rudeanu nr. 21, sector 1, București

Tel: 222.19.49, 223.19.28; Fax: 222.40.34; E-mail: corint@dnt.ro

Magazinul virtual al editurii Corint: www.edituracorint.ro

Toate drepturile asupra acestei ediții sunt rezervate
Editurii CORINT

Format: 16/70x100. Coli tipo: 12

Tipar executat la S.C. TIPARG S.A.

Pitești, B-dul N. Bălcescu, nr.93

Tel.: 0248-221 275; Fax: 0248-221 348

E-mail: tiparg@geostar.ro

MAI APROAPE DE COPII, MAI APROAPE DE CALCULATOR

An de an, copiii atrași de programarea calculatorului participă în număr tot mai mare la concursurile locale sau naționale de acest gen. De fiecare dată, participanții sunt invidiați de cei rămași fără „locuri” (calculatoare). Cei mai buni ridică ștacheta pe an ce trece, tot mai sus.

De la ce vârstă învață copiii să programeze și cum? Până unde poate fi ridicată ștacheta? Nu ne referim la limite, ci la capacități. Acestea pot fi dezvoltate. Luat în parte, fiecare copil are propria lui performanță. Programarea calculatorului este o provocare. Cu fiecare program izbutit, această provocare crește. Crește și încrederea în sine, dar și șansa de a fi mai bun în orice împrejurare. Copiii au nevoie de încurajare și recunoaștere. Calculatorul poate fi un partener.

Prin anii '70, matematicianul Grigore Moisil, vizionar cu har, anticipa era calculatoarelor prin cuvinte memorabile ca acestea: „Ce-ar fi ca la fiecare calculator să se înființeze o oră a copiilor?” La vremea respectivă, un calculator mobila o cameră, iar accesul se făcea prin „porți deschise” după un orar bine stabilit. În acest context, ora copiilor însemna timp de lucru la calculator rezervat în exclusivitate copiilor și aceasta era doar o dorință.

Privită din această perspectivă, cât de frumoasă pare situația astăzi! Nu calculatoarele au o oră a copiilor, ci copiii au și își doresc cât mai multe ore de calculator.

Chiar dacă școala nu oferă, încă, tuturor copiilor, această oră, există destui profesori, învățători, părinți pasionați care formează deprinderile de lucru cu calculatorul încă din ciclul primar. Foarte mulți copii desenează frumos la calculator, creează povești, „se plimbă” pe Internet, compun scenarii de site-uri fabuloase ori se pierd în lumea virtuală a jocurilor. Foarte mulți visează să creeze ei înșiși jocuri, să programeze.

Vârsta *copiilor programatori* coboară și ea spre ciclul primar. Cei mai mulți învață să programeze în gimnaziu.

Elementele de programare se regăsesc în tematica unor cursuri opționale din oferta școlii. În multe școli se organizează cercuri de programare, iar la cluburile copiilor din întreaga țară cursurile de acest gen sunt printre cele mai solicitate.

Experiența didactică și practica concursurilor de programare la care participăm ca îndrumători, organizatori sau membri în comisiile de jurizare ne-au motivat demersul de a realiza această carte mai „altfel” decât manualele sau culegerile existente, mai aproape de copii, mai aproape de calculator. Vrem să le oferim copiilor o altă modalitate de a învăța să programeze: prin exemple și rezolvări de situații concrete.

Programarea înseamnă, pentru noi, un mod de gândire și o formă de exprimare. Încercăm să transmitem această semnificație și copiilor. De aceea, am organizat conținutul începând cu **modelarea gândirii algoritmice** (partea I) și am continuat cu inițierea copiilor în **arta de a programa** (partea a II-a). Am ales limbajul Pascal din considerente didactice, dar și pentru faptul că este limbajul de programare cel mai folosit în mediul gimnazial.

Prin aceste lecții practice, copiii sunt orientați, pas cu pas, să gândească rezolvarea problemelor cu calculatorul de la analiza datelor până la compunerea testelor care probează calitatea unui program. Au fost alese atât probleme clasice, cât și probleme sau situații noi, adaptate vârstei.

În prezentare s-a urmărit și calitatea „produsului” realizat de programator: designul, ușurința în utilizare, performanța sau eficiența în timp a programului.

Conținutul este *accesibil* elevilor de gimnaziu atât prin nivelul cunoștințelor prezentate, cât și prin stilul abordat, bazat pe comunicare directă. Totodată, conținutul este și *provocator*, făcând apel la intuiție și creativitate.

Fiecare temă are un sumar de idei, sfaturi practice sau trucuri utile unui programator.

Pentru verificarea abilităților de gândire sau de programare dobândite, există algoritmi propuși spre descifrare sau programe care așteaptă să fie depanate.

Această lucrare este și un îndrumar pentru profesori, orientându-i în proiectarea și pregătirea scenariilor de lecții și poate fi parcursă în secvența partea I, partea a II-a, prin corelarea temelor din partea I cu cele din partea a II-a, sau selectiv, în funcție de nivelul de cunoștințe și interesul elevilor.

Autorii

PARTEA I

MODELAREA GÂNDIRII ALGORITMICE

„Gândirea algoritmică se insinuează pe neobservate în comportamentul nostru, în logica noastră, în modul nostru de a vedea lumea”
(Solomon Marcus)

1. Cine muncește mai mult: noi sau calculatorul?

Vrem să rezolvăm o problemă cu calculatorul. Ce avem de făcut?

La început, vom lucra *singuri, fără calculator*. Parcurgem următorii pași:

P1. Citim cu atenție enunțul problemei – trebuie să stabilim și să înțelegem cerința.

P2. Determinăm elementele cunoscute și elementele necunoscute din problemă – ceea ce „se dă” și ceea ce „se cere” sau datele de intrare și datele de ieșire.

P3. Notăm elementele problemei cu litere sau cuvinte sugestive pentru a stabili mai ușor condițiile sau relațiile dintre ele.

P4. Formulăm exemple cu valori concrete și rezolvăm problema pentru fiecare caz în parte. Reținem situațiile pentru care problema are o rezolvare specială sau nu are soluție.

P5. Stabilim un raționament general de rezolvare a problemei, valabil pentru toate cazurile.

P6. Reprezentăm raționamentului într-o formă cât mai simplă și clară.

P7. Verificăm raționamentul pentru valori concrete.

P8. Codificăm raționamentul în instrucțiunile limbajului de programare și obținem un **program**.

Din acest moment, avem nevoie de *calculator*.

Ce facem noi

P9. Transmitem calculatorului programul prin editare de text și comenzi pentru lucrul cu fișiere.

P10. Transmitem comanda de compilare (COMPILE).

P11. Corectăm și recompilăm programul până când compilatorul nu mai găsește nici o eroare.

P12. Transmitem comanda de execuție (RUN) și introducem valori pentru datele de intrare.

Ce face calculatorul

Înregistrează programul într-un **fișier sursă**.

Compilează programul: verifică corectitudinea sintactică a programului și semnalează erorile.

Execută programul instrucțiune cu instrucțiune și afișează pe ecran rezultatele (**datele de ieșire**).

P13. Verificăm soluțiile problemei pentru exemple concrete, cât mai variate.

P14. Comparăm rezultatele obținute de calculator cu cele obținute prin rezolvarea „de mână”. Când rezultatele diferă, e semn că s-a strecurat o greșeală în program sau chiar în raționament. În acest caz, o luăm de la capăt!

Fiecare pas din traseul pe care l-am prezentat va fi realizat în funcție de dificultatea sau complexitatea problemei. Schema generală a etapelor de rezolvare a problemelor cu calculatorul este prezentată în figura 1.

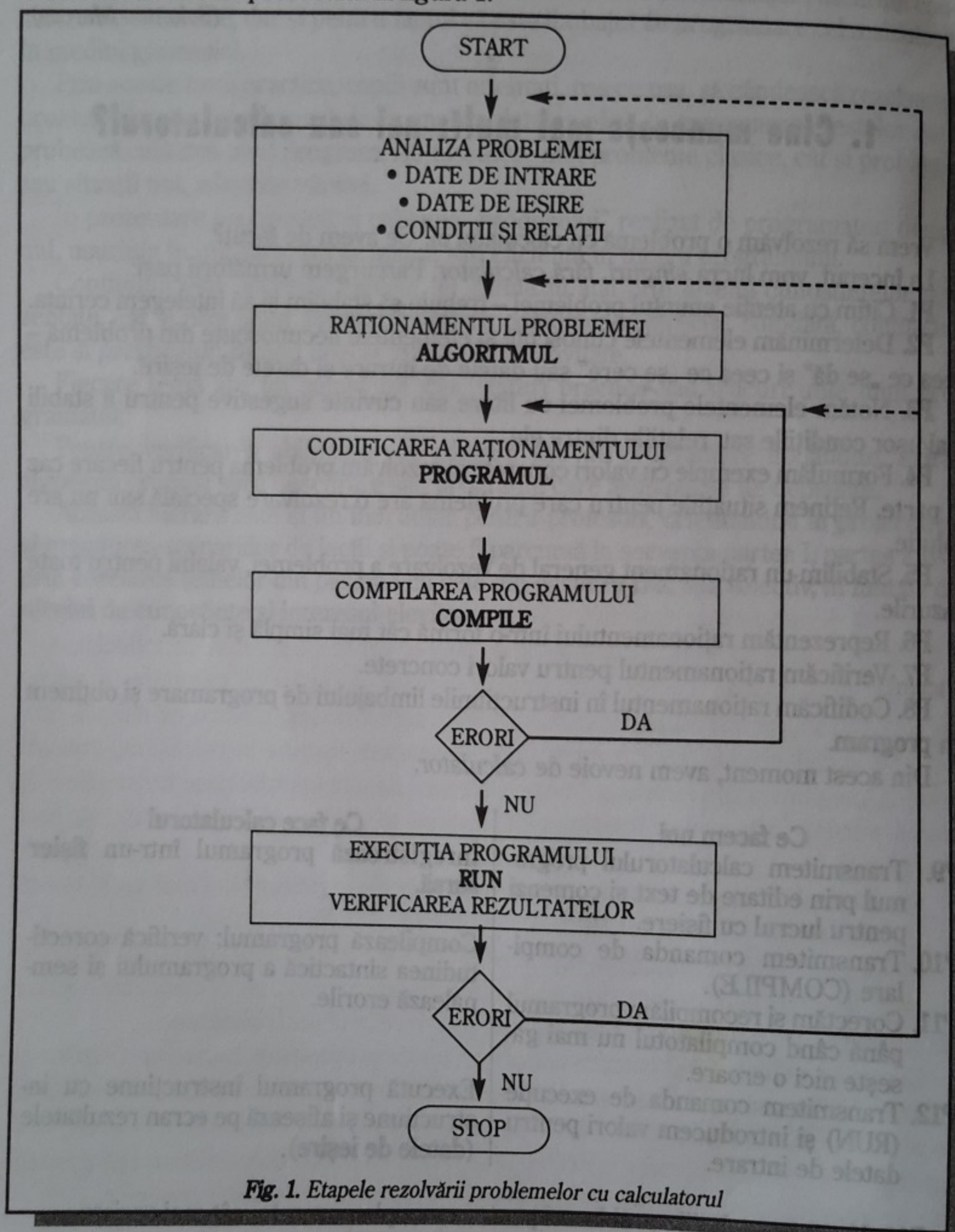


Fig. 1. Etapele rezolvării problemelor cu calculatorul

ZooCarnaval

La teatrul de păpuși se pregătește un spectacol nou. Costumierii trebuie să confecționeze pălării, pantofi și mănuși pentru o mulțime de personaje, care sunt păsări și animale. Păsările vor purta pălării și pantofi, iar animalele, pălării, mănuși și pantofi. Știind câte păsări și câte animale participă la carnaval, să se determine numărul de pălării și numărul de mănuși și pantofi pe care le vor realiza costumierii.

Să parcurgem împreună primii 5 pași spre rezolvarea problemei.

P1. Stabilirea cerinței: să se determine câte pălării, câți pantofi și câte mănuși vor realiza costumierii sau să se determine câte pălării și câți pantofi și mănuși la un loc vor realiza costumierii?

Citim cu atenție problema: în enunț se specifică „numărul de pălării” (câte?) și „numărul de mănuși și pantofi” (câți pantofi și mănuși la un loc?).

P2. Ce „se dă”:

- numărul de păsări;
- numărul de animale.

Ce „se cere”:

- numărul de capete (pălării);
- numărul de picioare (mănuși și pantofi).

P3. Notăm elementele problemei cu:

- x – numărul de păsări
- y – numărul de animale
- z – numărul de capete
- u – numărul de picioare

Stabilim relațiile dintre elementele problemei:

$$z = x + y$$

$$u = 2 * x + 4 * y$$

(1)

P4. Exemple și cazuri speciale:

a) la carnaval participă 5 păsări și 7 animale

$$z = 5 + 7 = 12 \text{ capete // pălării}$$

$$u = 2 * 5 + 4 * 7 = 38 \text{ picioare // mănuși și pantofi}$$

b) la carnaval participă doar păsări ($y = 0$)

$$z = x$$

$$u = 2 * x$$

c) la carnaval participă doar animale ($x = 0$)

$$z = y$$

$$u = 4 * y$$

(2)

(3)

P5. Raționamentul problemei

Se introduc valori pentru datele de intrare (x și y).

Se aplică relațiile (1) și se obțin datele de ieșire (z și u).

Întrucât relațiile (2) și (3) se obțin din (1) raționamentul rezolvă toate cazurile.

Ce trebuie să știm dacă vrem să rezolvăm o problemă cu calculatorul

1. Calculatorul nu gândește în locul nostru.
2. Calculatorul nu gândește cum vrem noi, ci așa cum îl învățăm noi.
3. Înainte de a învăța calculatorul să gândească, trebuie să învățăm noi înșine să gândim corect.

2. Și calculatorul învață!

Creierul electronic al calculatorului rezolvă orice problemă dacă este corect instruit sau „învățat” să facă acest lucru. „Profesorul” calculatorului este *programatorul*.

Programarea nu înseamnă doar „traducerea” unui raționament într-un limbaj specific calculatorului – *limbajul de programare* – ci, mai ales, compunerea raționamentului. Programatorul construiește un întreg scenariu logic după care să gândească sau să lucreze calculatorul. Acest scenariu necesită rațiune, dar și imaginație, intuiție, creativitate. De aici vine și expresia „arta gândirii logice” sau „arta programării”.

Mulți începători sau „amatori” în programare consideră că însușirea unui set de instrucțiuni specifice unui limbaj de programare este o țintă în sine, o performanță. Limbajul de programare poate fi comparat cu o limbă străină. El ne ajută să comunicăm în acest caz, cu calculatorul. Ce comunicăm? În această situație – un raționament adus la forma unei secvențe exacte de operații logice de gândire numită **algoritm**. Orice program modelează, exprimă, comunică calculatorului un algoritm.

Deși foarte modern, fiind specific astăzi informaticii, termenul de algoritm provine dintr-o formă mai veche, **algorism**, care însemna procesul efectuării operațiilor aritmetice folosind cifrele arabe. Denumirea în sine, ca și aceea de algebră, este legată de numele matematicianului arab Abu Ja Far Mohammed Ibn Mûsâ al Khowarismî și de lucrarea sa Kitab al-jabr wa'l-muqâbala scrisă în anul 825 d.Hr. și în care sunt tratate operațiile asupra numerelor.

În vorbirea curentă, algoritmul reprezintă un proces sau procedeu prin care se rezolvă în amănunt o problemă oarecare, nu neapărat de matematică.

Pentru un programator, algoritmul are semnificații și proprietăți specifice modelării gândirii electronice a calculatorului.

Algoritmul este o succesiune (secvență) de operații elementare de gândire. Calculatorul doar execută, de aceea trebuie să gândim în amănunțime fiecare operație care duce la rezolvarea problemei.

Pentru un elev de clasa a V-a este „evident” că 2, 4, 86 etc. sunt numere pare. Calculatorul „știe” să deosebească un număr par de unul impar? Începătorii vor fi dezamăgiți că prietenul lor, megacalculatorul nu știe „atâta lucru”.

Ce „știe” de fapt, calculatorul? În general, orice calculator recunoaște prin instrucțiunile limbajului de programare, operațiile aritmetice elementare – adunare,

scădere, înmulțire, împărțire întreagă (câtul sau restul) și zecimală și operația logică de a compara între ele cele două valori.

Cum „știe” calculatorul dacă un număr este par sau nu? Îl vom învăța noi, după ce vom compune algoritmul corespunzător: transmitem calculatorului numărul care ne interesează, fie acesta x . Calculatorul îl împarte pe x la 2 și, dacă restul împărțirii este zero, afișează pe ecran mesajul „număr par” altfel, afișează mesajul „număr impar”.

Algoritmul este finit. Calculatorul execută instrucțiunile transmise prin program și, la un moment dat, va trebui să se oprească, să termine ceea ce a început și să prezinte rezultatele.

O problemă de genul „Să se determine toate puterile unui număr întreg” x nu poate fi rezolvată de nici un calculator. Secvența valorilor cerute este de forma $x, x^2, x^3, x^4, x^5, \dots$ și reprezintă o secvență fără sfârșit, **infinită**, care nu poate fi algoritmicizată. Deși valorile se obțin simplu prin înmulțirea repetată cu x , nu există o condiție de oprire. Până când va repeta calculatorul înmulțirea? Sau cât timp? În acest caz, problema trebuie reformulată prin introducerea unei condiții.

De exemplu: Să se determine toate puterile, **mai mici decât 1000**, ale unui număr întreg oarecare x .

Pentru această cerință se poate formula un algoritm: „se repetă înmulțirea lui x cu el însuși, până când se obține o valoare mai mare decât 1000”.

În acest caz, secvența operațiilor efectuate de calculator este numărabilă, adică finită.

Algoritmul este general – specific unei mulțimi de probleme de același fel (clasă de probleme).

Rezolvarea unei probleme de genul „să se determine perimetrul unui triunghi cu laturile de 3 cm, 4 cm și 5 cm” nu necesită un algoritm, întrucât formularea este pentru un caz particular, cu valori concrete. Reformulăm problema: „Ne propunem să determinăm perimetrul oricărui triunghi cu laturile egale cu valori oarecare de a cm, b cm, c cm”. În acest caz, rezolvarea problemei conduce la un algoritm întrucât, oricare ar fi triunghiul dat, va exista o soluție proprie calculată după relația

$$\text{perimetru} = a + b + c.$$

Trei reguli de bază pentru compunerea unui algoritm

1. **Să despici firul în patru:** să analizezi problema sub toate aspectele și să gândești rezolvarea ei până în cele mai mici amănunte.
2. **Să termini întotdeauna lucrul început:** să nu pui calculatorul să execute operații fără sens (imposibile) sau fără sfârșit.
3. **Să cauți soluții pentru toate cazurile problemei:** un lucru bine făcut, poate fi folosit de oricine, oricând.

3. Memoria – o resursă limitată

Deși nu este o „mașinărie”, putem spune și despre un algoritm că „funcționează” întrucât „ceva” se schimbă de la o operație la alta, în urma unor „acțiuni”. Acest „ceva” despre care vorbim este reprezentat de datele problemei, iar „acțiunile” sunt prelucrările aritmetico-logice aplicate acestor date.

Privit astfel, ca un mecanism, algoritmul transformă *datele de intrare* ale problemei în *date de ieșire*. Transformarea efectivă are loc în timpul execuției programului.

Datele-valori concrete specifice problemei vor fi reținute de calculator în zone de memorie solicitate prin instrucțiuni specifice limbajului de programare. Aceste zone de memorie rămân la dispoziția programului cât timp durează execuția acestuia. Memoria calculatorului este o resursă limitată care trebuie folosită cu măsură. Programatorul are sarcina de a se gândi atât la algoritmul de rezolvare a problemei, cât și la folosirea eficientă a memoriei.

Intuitiv, memoria calculatorului poate fi asemănată cu o suprafață formată din puncte (locații) învecinate (adiacente). Fiecare locație are o adresă prin care poate fi identificată și accesată. Mai multe locații formează o zonă de memorie cu o dimensiune sau capacitate bine definită.

Spre exemplu, pentru determinarea perimetrului P al unui triunghi cu laturile A , B , C sunt necesare patru zone de memorie (figura 2).

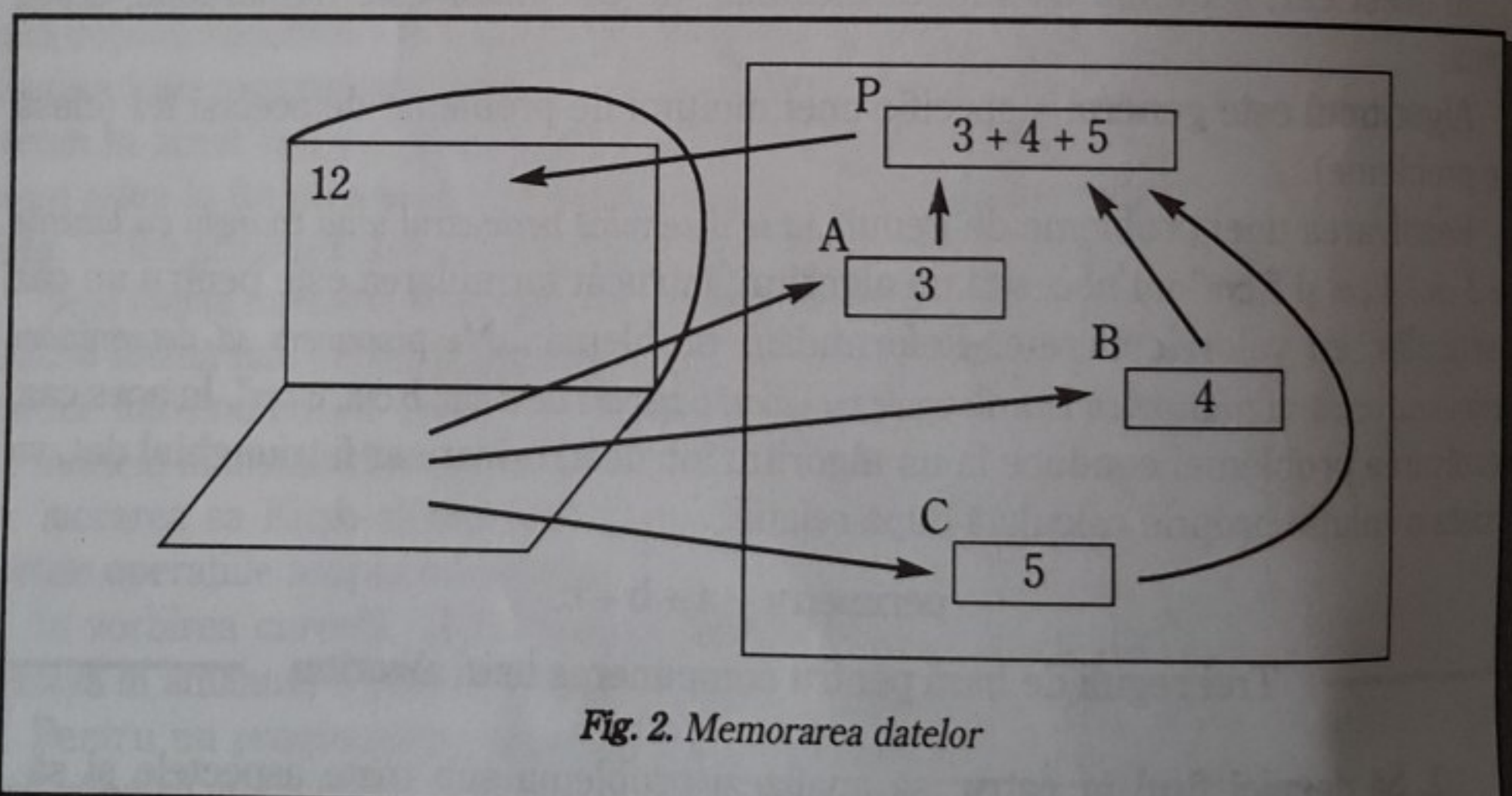


Fig. 2. Memorarea datelor

Pentru calculator, valorile sau datele unei probleme sunt informații codificate prin doar două valori (0 și 1) specifice sistemului **binar** (baza de numerație 2).

Dimensiunea sau capacitatea zonelor de memorie se exprimă în unități de măsură specifice informației. Cea mai mică unitate de măsură pentru informație este **bitul**. Să exprimi capacitatea de memorie în biți este ca și cum ai măsura Pământul în milimetri!

Multipli bitului sunt: 1 Byte (B) = 8 biți = 1 octet
1 KiloByte (KB) = 2^{10} B = 1024 B
1 MegaByte (MB) = 2^{10} KB
1 GigaByte (GB) = 2^{10} MB

Reguli pentru utilizarea memoriei

1. Datele problemei sunt memorate în zone de memorie care pot fi adresate din program (algoritm) prin numele asociate datelor.
2. O zonă de memorie păstrează la un moment dat o singură valoare.
3. Valoarea reținută într-o zonă de memorie poate fi modificată prin operații matematice.
4. Conținutul unei zone de memorie poate fi copiat (transmis) în altă zonă de memorie.
5. Conținutul unei zone de memorie poate fi comparat cu altă zonă de memorie.
6. O zonă de memorie poate primi valori prin citire direct de la tastatură.
7. Conținutul unei zone de memorie poate fi afișat prin scriere pe ecranul calculatorului.

4. Time out!

Pentru transformarea datelor de intrare în date de ieșire, algoritmul necesită spațiu de memorie și timp de execuție. Fiecare operație consumă timp măsurat de ceasul intern al calculatorului în diviziuni de secundă.

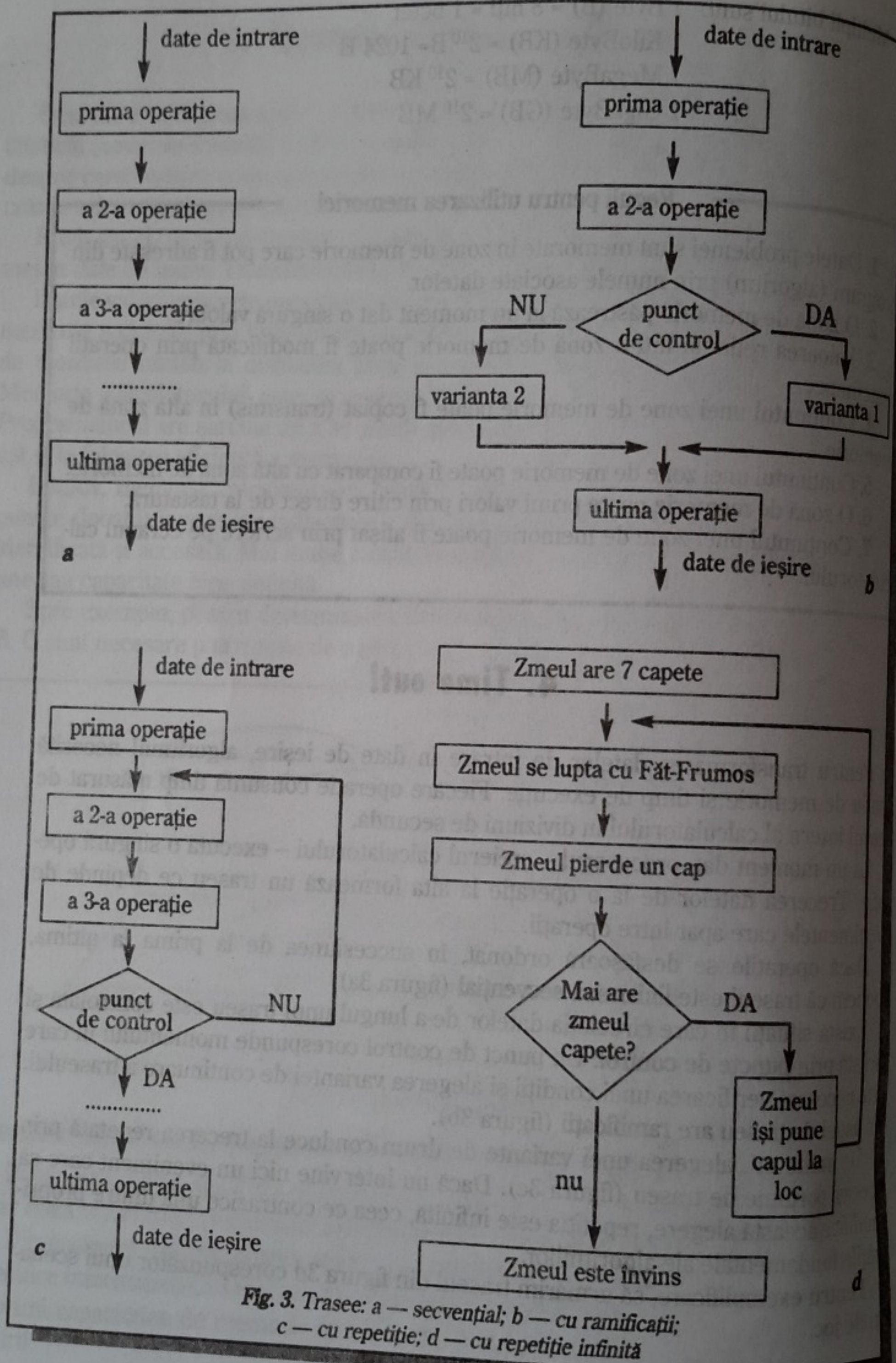
La un moment dat, procesorul – creierul calculatorului – execută o singură operație. Trecerea datelor de la o operație la alta formează un traseu ce depinde de evenimentele care apar între operații.

Dacă operațiile se desfășoară ordonat, în succesiunea de la prima la ultima, spunem că traseul este **liniar** sau **secvențial** (figura 3a).

Există situații în care circulația datelor de-a lungul unui traseu este controlată și dirijată prin **puncte de control**. Un punct de control corespunde momentului în care este necesară verificarea unei condiții și alegerea variantei de continuare a traseului. Un astfel de traseu are **ramificații** (figura 3b).

De multe ori, alegerea unei variante de drum conduce la trecerea repetată prin aceeași porțiune de traseu (figura 3c). Dacă nu intervine nici un eveniment care să modifice această alegere, repetiția este infinită, ceea ce contrazice una dintre proprietățile fundamentale ale algoritmilor.

Pentru exemplificare, să urmărim traseul din figura 3d corespunzător unui scenariu de joc.



Printre operații (secvențele de scenă ale jocului) există două al căror efect contradictoriu face ca, în punctul de control, condiția ca zmeul să nu mai aibă capete să nu fie niciodată îndeplinită. Jocul nu are sfârșit.

Chiar există algoritmi infiniti? Dar programe? Algoritm infinit nu există. Repetiția infinită contrazice definiția și proprietățile algoritmilor. Dacă un algoritm conține o repetiție infinită, acela este un algoritm greșit, prost construit și trebuie corectat.

Programele pot fi însă fără sfârșit. Dacă printr-o eroare de programare un algoritm este greșit codificat și se formează o repetiție infinită, teoretic programul nu se mai oprește. Calculatorul este pregătit să controleze și astfel de situații: fiecare program are la dispoziție un interval limită de timp în care, în mod normal, ar trebui să se termine. Dacă limita de timp este depășită, programul este întrerupt și abandonat forțat cu un mesaj sever: *time out!*

Reguli pentru compunerea unui algoritm corect

1. Orice algoritm are o singură operație inițială și o singură operație finală.
2. Locul fiecărei operații în succesiune este fix, nu poate fi modificat în timpul execuției.
3. Ordinea de execuție a operațiilor dirijează fluxurile de date de la datele de intrare spre datele de ieșire.
4. Dirijarea fluxurilor de date se face prin puncte de control în care sunt verificate condițiile problemei.
5. Înainte de a codifica algoritmul, se verifică fluxurile de date: să nu treacă prin circuite sau bucle fără ieșire (infinite).

5. POVESTE PE GUSTUL FIECĂRUIA

după Raymond Queneau

1. Doriți să cunoașteți povestea celor trei mici boabe de mazăre?
Dacă DA, mergeți la 4.
2. Sau poate o preferați pe cea a celor trei mici castraveciori subțiri?
Dacă DA, mergeți la 16.
3. Vă plac poate mai mult aventurile celor trei copăcei de 3 ani?
Dacă DA, mergeți la 17; dacă NU, mergeți la 20.
4. Erau odată trei boabe de mazăre mici care dormeau liniștite în culcușul lor verde. Fața lor rotundă era lucioasă, respirația egală și nu se auzea decât un sforăit ușor, armonios.
Dacă preferați altă descriere, mergeți la 9.
5. Ele nu visau. Aceste mici ființe nu visează niciodată.
Dacă preferați ca ele să viseze, mergeți la 6; dacă nu, mergeți la 7.

6. Ele visau. Aceste mici ființe visează totdeauna și nopțile lor sunt pline de vise încântătoare.

Dacă vreți să le cunoașteți visele, mergeți la 11.

7. Picioarele lor mici erau învelite, iar mâinile purtau mănuși de velur negru.

Dacă vă convine culoarea mănușilor, mergeți la 10.

8. Ele purtau în pat mănuși de velur albastru.

Dacă preferați mănuși de altă culoare, interesați-vă unde se poate găsi velur de acea culoare și anunțați-ne și pe noi; dacă aceasta este culoarea care vă convine, mergeți la 10.

9. Erau odată trei mici boabe de mazăre care se rostogoleau pe marginea drumului. O dată cu căderea serii, obosite de atâta mers, au adormit foarte rapid.

Dacă doriți să cunoașteți urmarea, mergeți la 5; dacă NU, mergeți la 20.

10. Toate trei visau același vis. Fiind din aceeași păstaie, deci gemene, toate gândurile și visele lor erau aceleași.

Dacă nu vă interesează ce visau, mergeți la 12.

11. Ele visau că erau invitate la o masă unde li s-a servit ca prim fel supă de linte. Îngrozite de această situație, s-au trezit.

Dacă vreți să știți de ce le-a speriat situația, luați dicționarul și căutați la cuvântul „linte”. Nu are rost să mai continuăm.

Dacă însă vi se pare inutil să aprofundați problema, mergeți mai departe.

12. „Opopoo!” strigară ele deschizând ochii; „Opopoo! ce coșmar am putut să avem!” „Semn rău!” a zis prima. „Așa este” a completat a doua; „O să fim triste toată ziua!”. „Nu vă tulburați atâta” i-a liniștit a treia boabă de mazăre, care era totuși mai coaptă. „Nu trebuie să vă speriați, ci să încercați să înțelegeți; haideți să facem o analiză”.

Dacă vreți să aflați interpretarea visului, mergeți la 15. Dacă vă interesează și reacția celorlalte două boabe de mazăre, continuați.

13. „Te dai mare!” a ripostat prima. „De când știi tu să tălmăcești visele?” „Așa este! De când?” a adăugat a doua boabă de mazăre.

Dacă vreți să știți și dumneavoastră de când, mergeți mai departe; dacă NU vă interesează acest amănunt, mergeți totuși mai departe, că tot nu veți afla mai mult.

14. „De când?” s-a întrebat a treia boabă de mazăre. „Habar n-am; dar cert este că pot interpreta visul. Să vedeți:”

Dacă vreți și dumneavoastră să vedeți, mergeți mai departe; dacă NU, mergeți mai departe totuși, că tot nu veți vedea nimic.

15. „Ei bine, dacă tu ești așa dotată, să te auzim!” au cedat surorile ei.

„Ironia voastră nu-mi face plăcere” s-a supărat a treia boabă de mazăre. „Așa că nu vă mai spun nimic. De altfel, de când vorbim, nu v-a plăcut senzația de spaimă? La ce bun să mai răscolim lucrurile! Haideți mai bine la fântână să ne spălăm și să salutăm cum se cuvine soarele care tocmai a răsărit.” Și, liniștite, cele 3 boabe de mazăre așa au și făcut: au ieșit rostogolindu-se din păstaia lor de culcuș, s-au rostogolit spre apă și curând tot locul răsună de activitatea lor veselă.

Dacă doriți să știți care era activitatea lor, mergeți mai departe; dacă NU mergeți la 20.

16. Trei castraveți mari priveau scena.

Dacă faptul că au crescut castraveții nu vă place, mergeți la 20; dacă vă convine, mergeți la 18.

17. Trei arbuști de 3 ani priveau scena.

Dacă scena sau arbuștii nu vă mai plac, mergeți la 20; dacă v-ați plictisit să mai interveniți, lăsați totul să meargă mai departe.

18. Văzându-se studiați așa de atent, protagoniștii s-au speriat și au fugit rostogolindu-se cât au putut de repede.

Dacă nu vă interesează unde au fugit, mergeți la 20.

19. Ei au fugit să se adăpostească în păstaia lor calduroasă și – o dată ajunși aici – au adormit din nou.

Dacă doriți să cunoașteți urmarea, mergeți la 5.

20. Cum orice poveste are un sfârșit mai devreme sau mai târziu, și această poveste s-a încheiat.

(Gazeta de Informatică 10/1993 pag. 28–29)

6. Reprezentarea algoritmilor

Ați reușit să citiți pe gustul vostru povestea celor trei boabe de mazăre sau trei castraveci sau trei copăcei de 3 ani?

Dacă DA, înseamnă că v-a plăcut; dacă NU, s-ar putea să fie vorba de gust sau poate v-ați plictisit sau v-ați încurcat în ițele poveștii. Prea multe fire și cam greu de urmărit!

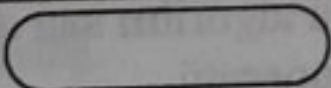
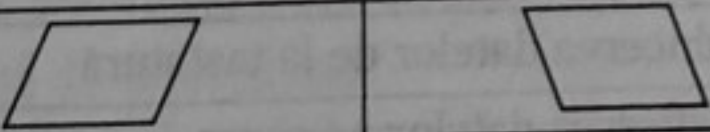
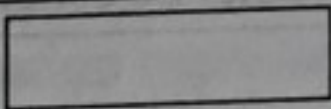
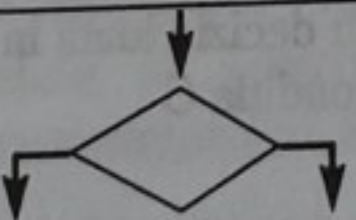
Fiind vorba despre o poveste, se acceptă prezentarea narativă. Cum vom prezenta însă un algoritm? Dacă îl vom „povesti” s-ar putea să ne încurcăm noi înșine în traseele încâlcite ale datelor.

Vom căuta o reprezentare cât mai ordonată, simplă și clară. Există două modalități de reprezentare a algoritmilor: reprezentarea prin **schemă logică** și reprezentarea prin **pseudocod**.

Schema logică este o reprezentare grafică prin simboluri (șabloane) cu semnificație bine precizată (vezi tabelul 1). Fiecare operație din algoritm se reprezintă prin șablonul corespunzător. Trecerea de la o operație la alta este sugerată prin săgeți.

Șabloanele schemei logice

Tabelul 1

Simbol grafic	Semnificație
	început sau sfârșit de algoritm
	citirea datelor scrierea datelor
	operație/prelucrare
	punct de control

Schema logică din figura 4 reprezintă algoritmul de rezolvare a problemei ZooCarnaval.

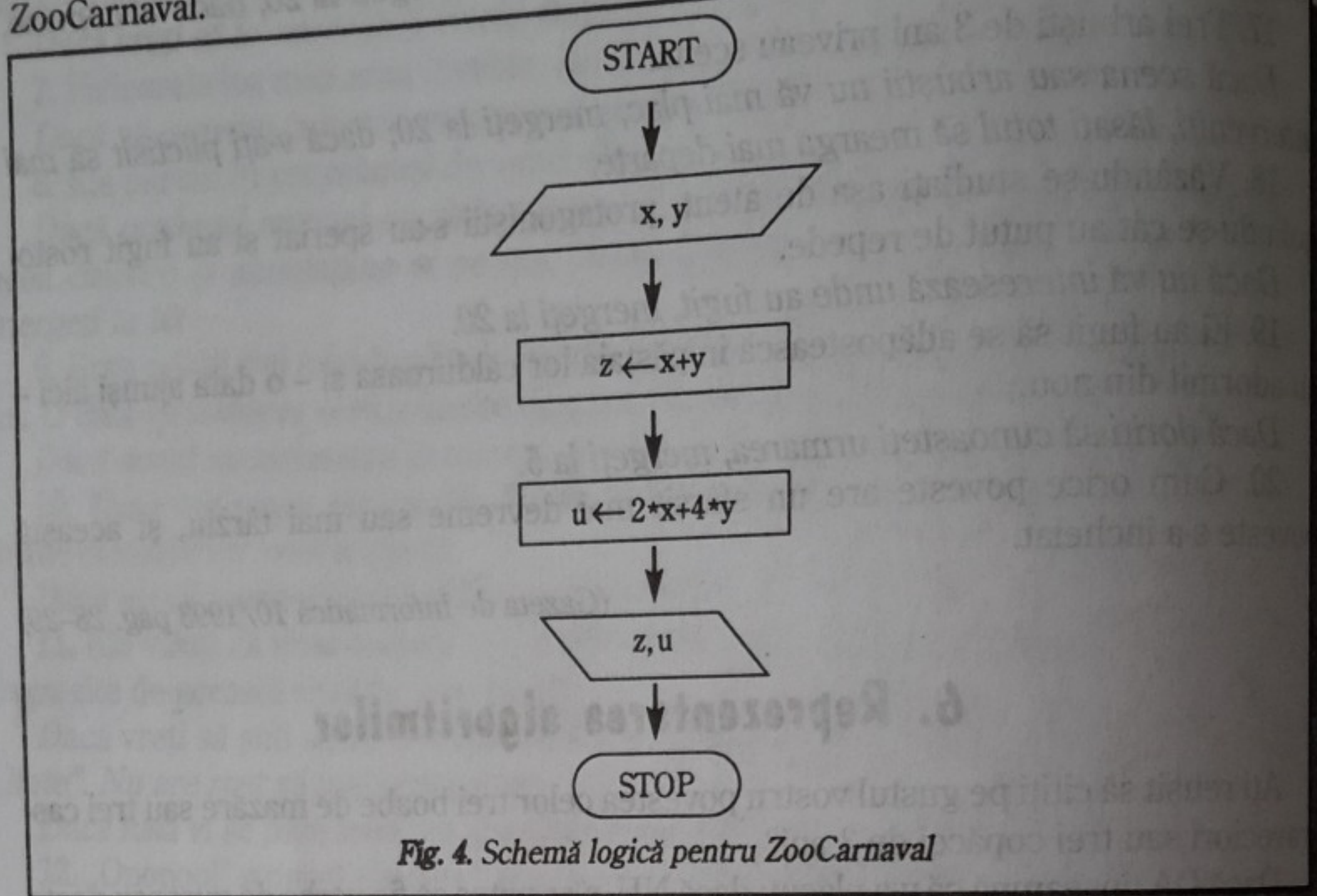


Fig. 4. Schemă logică pentru ZooCarnaval

Reprezentarea prin schemă logică necesită mai mult timp dar este foarte clară și ușor de urmărit. Fluxurile de date urmează traseele marcate prin săgeți de la **Start** până la **Stop**.

Reprezentarea prin **pseudocod** folosește un set de cuvinte. De aceea, această reprezentare este mai apropiată de modul nostru de gândire și, în același timp, de codificarea algoritmului într-un limbaj de programare.

Există mai multe variante de pseudocod în funcție de vocabularul folosit: cel al limbii engleze sau cel al limbii române. În această lucrare folosim pseudocod în limba română cu semnificația din tabelul 2 prezentat în continuare.

Pseudocod în limba română		Tabelul 2
Pseudocod	Semnificație	
început sfârșit	începutul/sfârșitul unui algoritm sau al unui bloc de operații	
citește	introducerea datelor de la tastatură	
scrie	afișarea datelor pe ecran	
dacă C atunci altfel sf dacă	punct de control sau decizie luată în funcție de condiția C	

Tabelul 2 (continuare)

Pseudocod	Semnificație
repetă până când C	secvențe repetitive condiționate
cât timp C execută sf cât timp	
pentru contor=vi, vf execută sf pentru	secvență repetitivă cu contor
DIV	câtul împărțirii a două numere întregi
MOD	restul împărțirii a două numere întregi
←	atribuire
//	comentariu

Exemple de reprezentări pseudocod:

a) algoritmul problemei ZooCarnaval

b) algoritm pentru determinarea proprietății de număr par

început zoo

 citește x

 citește y

$u \leftarrow x + y$

$z \leftarrow 2 * x + 4 * y$

 scrie u

 scrie z

sfârșit zoo

început par

 citește x

$r \leftarrow x \bmod 2$

 dacă $r = 0$

 atunci scrie x, "par"

 altfel scrie x, "impar"

 sf dacă

sfârșit par

Cum reprezentăm algoritmii

1. Un programator iscusit trebuie să cunoască atât reprezentarea în schemă logică, cât și reprezentarea în pseudocod.
2. Pentru probleme foarte simple, algoritmul poate fi reprezentat direct în limbaj de programare.
3. Programatorul va decide forma de reprezentare a algoritmului în funcție de experiență sa și de dificultatea problemei.
4. Reprezentarea în schemă logică este mai ușor de urmărit și pune în evidență fluxurile de date.
5. Reprezentarea în pseudocod folosește un set redus de cuvinte ușor de reținut și de interpretat.
6. Pseudocodul nu este accesibil calculatorului.
7. Un algoritm reprezentat în schemă logică sau pseudocod este ușor de codificat în orice limbaj de programare.
8. Limbajele de programare au instrucțiuni corespunzătoare elementelor grafice sau cuvintelor pseudocod.

7. Să gândim structurat

Nu știu câți dintre noi s-ar încumeta să reprezinte în schemă logică sau pseudocod povestea ... „pe gustul fiecăruia”.

Un algoritm este ușor de reprezentat dacă operațiile ce compun raționamentul problemei sunt corect organizate în grupuri numite **structuri**.

O structură poate fi imaginată ca o cutie cu două capace: unul se ridică pentru a pune ceva în cutie (punct de intrare), celălalt, pentru a scoate ceva din cutie (punct de ieșire). Ce intră și ce iese din cutie (structură)? Datele problemei.

Pe tot parcursul transformării lor din date de intrare în date de ieșire, fluxurile de date intră și ies dintr-o structură în alta.

Oricare ar fi raționamentul de rezolvare al unei probleme, operațiile aritmetico-logice pot fi grupate în doar trei categorii de structuri:

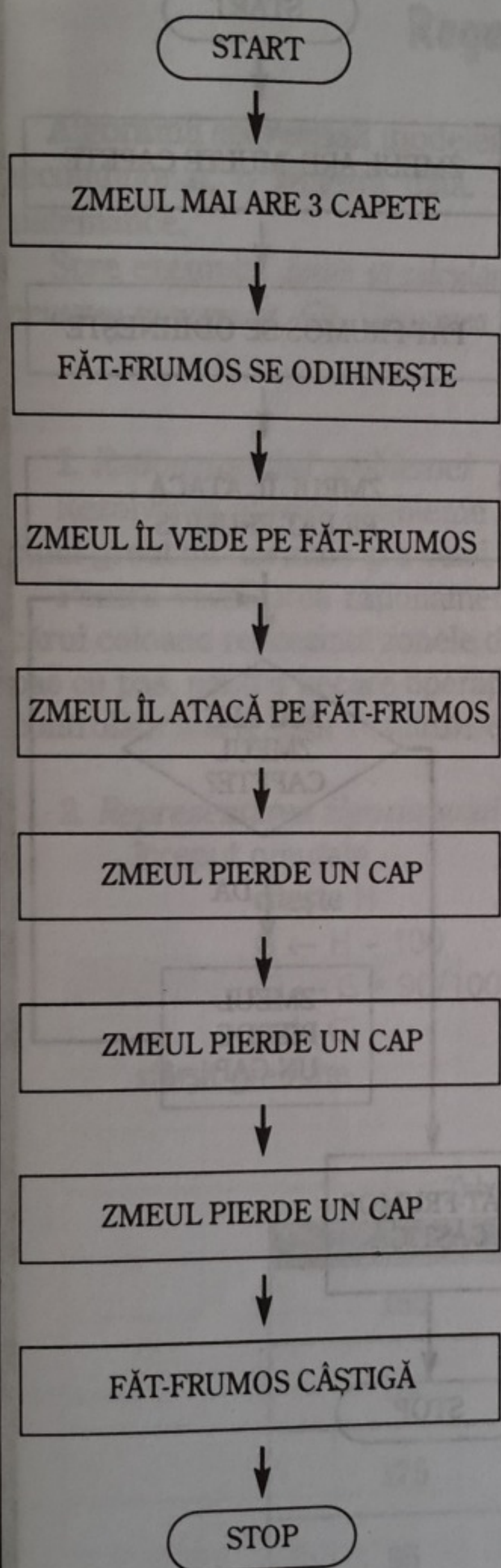
- **structuri liniare** compuse din una sau mai multe operații care se execută secvențial în ordinea de la prima la ultima;
- **structuri cu ramificație** compuse dintr-un punct de control și două sau mai multe ramificații spre operații care se execută în funcție de condiția verificată;
- **structuri repetitive** compuse din una sau mai multe operații a căror repetiție este controlată printr-un punct de control.

Tehnica de compunere a algoritmurilor și a programelor folosind doar aceste tipuri de structuri se numește **gândire** sau **programare structurată**.

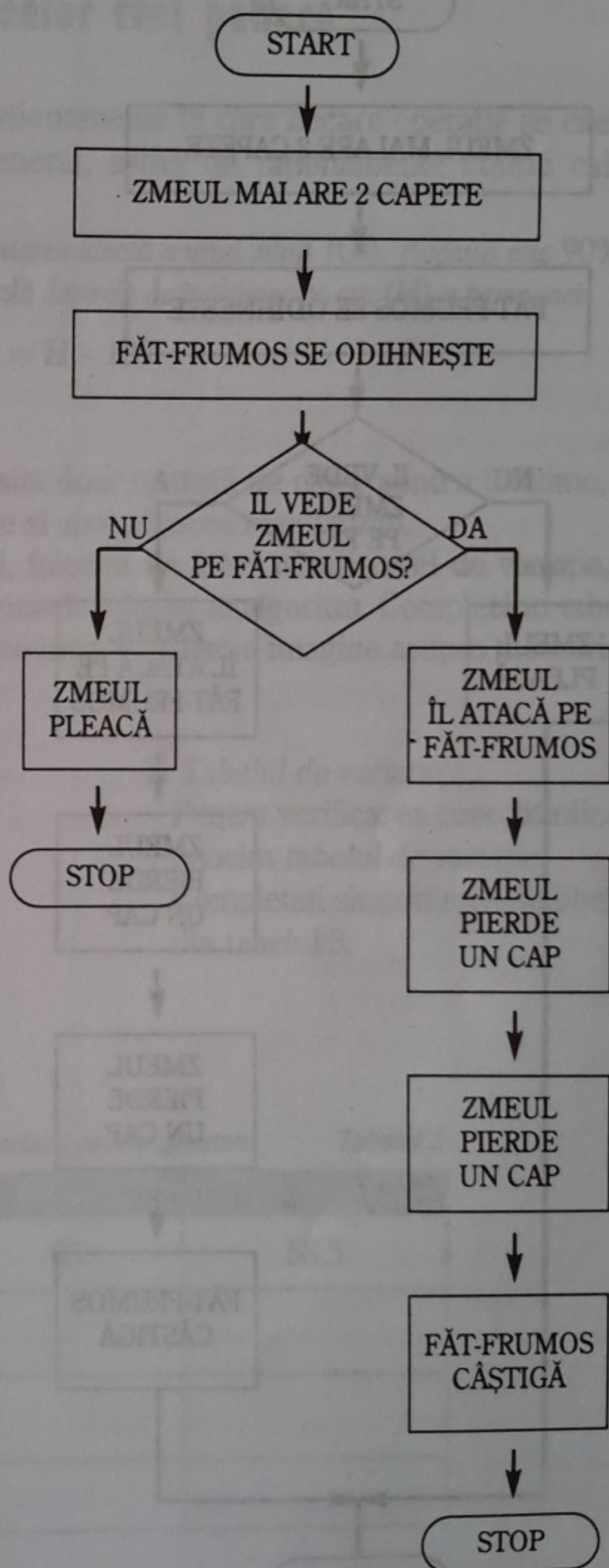
În figura 5 sunt prezentate aceste structuri pe exemplul scenariului de joc „FFZ” (Făt-Frumos și Zmeul). Cadrele jocului sunt organizate în structuri liniare (secvențiale) în figura 5a. Introducerea în scenariu a condiției „îl vede zmeul pe Făt-Frumos?” necesită aranjarea cadrelor în structura cu două ramificații (decizională) din figura 5c. Reprezentarea din figura 5b nu este corectă deoarece structura decizională are două ieșiri. Cele patru operații reprezentate secvențial în figura 5a sunt grupate în structura repetitivă din figura 5d.

Cum gândim structurat

1. Grupăm în structuri secvențiale operațiile a căror execuție nu este condiționată de nici un eveniment.
2. Grupăm în structuri cu ramificație operațiile a căror execuție depinde de îndeplinirea unei condiții (structuri decizionale) sau de valoarea unui selector (structuri de selecție).
3. Grupăm în structuri repetitive operațiile care se repetă.
4. Pentru fiecare dintre structuri există elemente corespunzătoare în schemă logică, pseudocod sau limbaj de programare.
5. Un algoritm corect structurat este ușor de codificat în limbaj de programare.



a



b

Fig. 5. Structuri: a — liniare (secvențiale); b — cu două ramificații (încorectă)

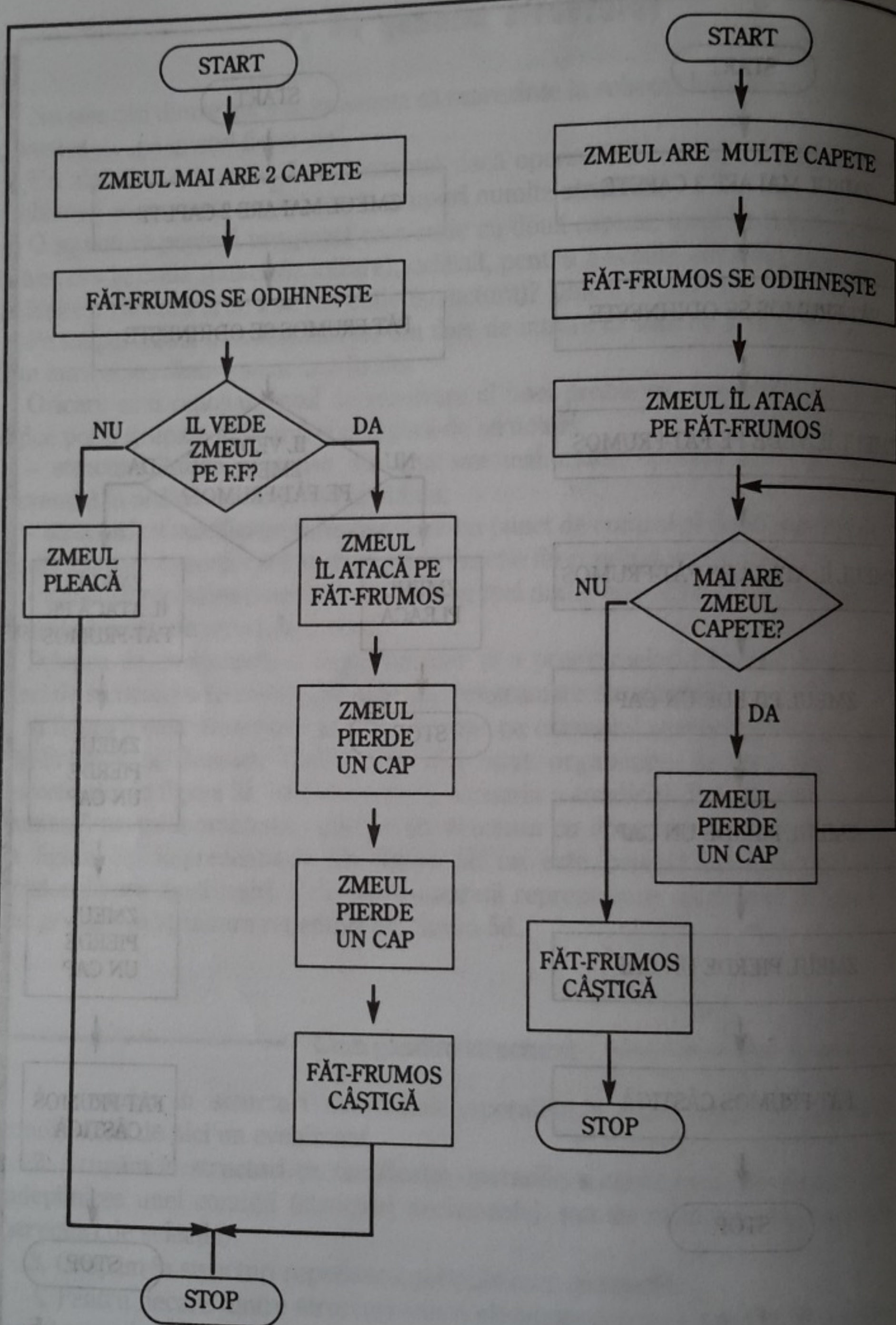


Fig. 5. Structuri: c — cu două ramificații (corectă); d — repetitive

8. Algoritmi secvențiali

Regula celor trei pahare

Algoritmii secvențiali modelează raționamente în care fiecare operație se execută necondiționat, o singură dată. În general, astfel de raționamente conțin calcule matematice.

Spre exemplu, dorim să calculăm greutatea ideală a unui adult (G_i). Aceasta este 90% din greutatea sa normală (G). Greutatea normală depinde de înălțimea în cm (H) a persoanei:

$$G = H - 100$$

1. Raționamentul problemei

Rezolvarea acestei probleme necesită doar operații de citire pentru înălțime, calculul greutății normale și a celei ideale și apoi afișarea rezultatului.

Pentru verificarea raționamentului, folosim un tabel numit **tabel de variație**, ale cărui coloane reprezintă zonele de memorie folosite în algoritm. Completând tabelul, pas cu pas, pentru fiecare operație executată, obținem o imagine asupra memoriei și controlăm foarte ușor rezultatele.

2. Reprezentarea algoritmului

început greutate
citește H
 $G \leftarrow H - 100$
 $G_i \leftarrow G * 90/100$
scrie G_i
sfârșit greutate

3. Tabelul de variație

Pentru verificarea corectitudinii folosim *tabelul de variație*. Completați singuri spațiile libere din tabelul 3.

Tabel de variație pentru greutate

Tabelul 3

H	G	G_i
165	65	58,5
182		
175		
95		

O astfel de rezolvare consideră datele de intrare corecte. Pentru înălțimea de 95 cm, care nu corespunde unui adult, greutatea ideală este un număr negativ lipsit de semnificație în contextul problemei.

În foarte mulți algoritmi este necesar transferul de valori între zone de memorie. Acest transfer se numește **interschimb**. Fie **A** și **B** cele două zone de memorie. În tabelul de variație sunt prezentate valorile acestor zone înainte și după interschimb:

A	B	
10	4	← valorile inițiale
4	10	← valorile după interschimb

1. Care este secvența de operații prin care se face interschimbul?

Dacă **A** și **B** ar fi două pahare pline și ar trebui să mutăm conținutul celor două pahare dintr-unul în altul, atunci fiecare dintre noi ar folosi cu siguranță, un al treilea pahar și ar proceda în felul următor (figura 6):

- prima operație: se varsă conținutul paharului **A** în paharul **C**;
- a doua operație: se varsă conținutul paharului **B** în paharul **A**;
- a treia operație: se varsă conținutul paharului **C** în paharul **B**.

Revenim la cele două zone de memorie și urmând „raționamentul celor trei pahare”, utilizăm o a treia zonă de memorie, **C** (figura 7). Spre deosebire de pahare, care se pot goli de conținut, se observă că „vărsarea” valorilor dintr-o zonă de memorie în alta nu se face prin „golire”, ci prin copiere sau atribuire.

Pentru realizarea unui algoritm complet, considerăm că cele două zone de memorie **A** și **B** primesc valori de la tastatură. După interschimb, valorile sunt afișate pe ecran.

2. Reprezentarea algoritmului

început interschimb1

citește A

citește B

$C \leftarrow A$

$A \leftarrow B$

$B \leftarrow C$

scrie A

scrie B

sfârșit interschimb1

A	B	C
10	4	
		10
4	10	
25	16	
9	32	

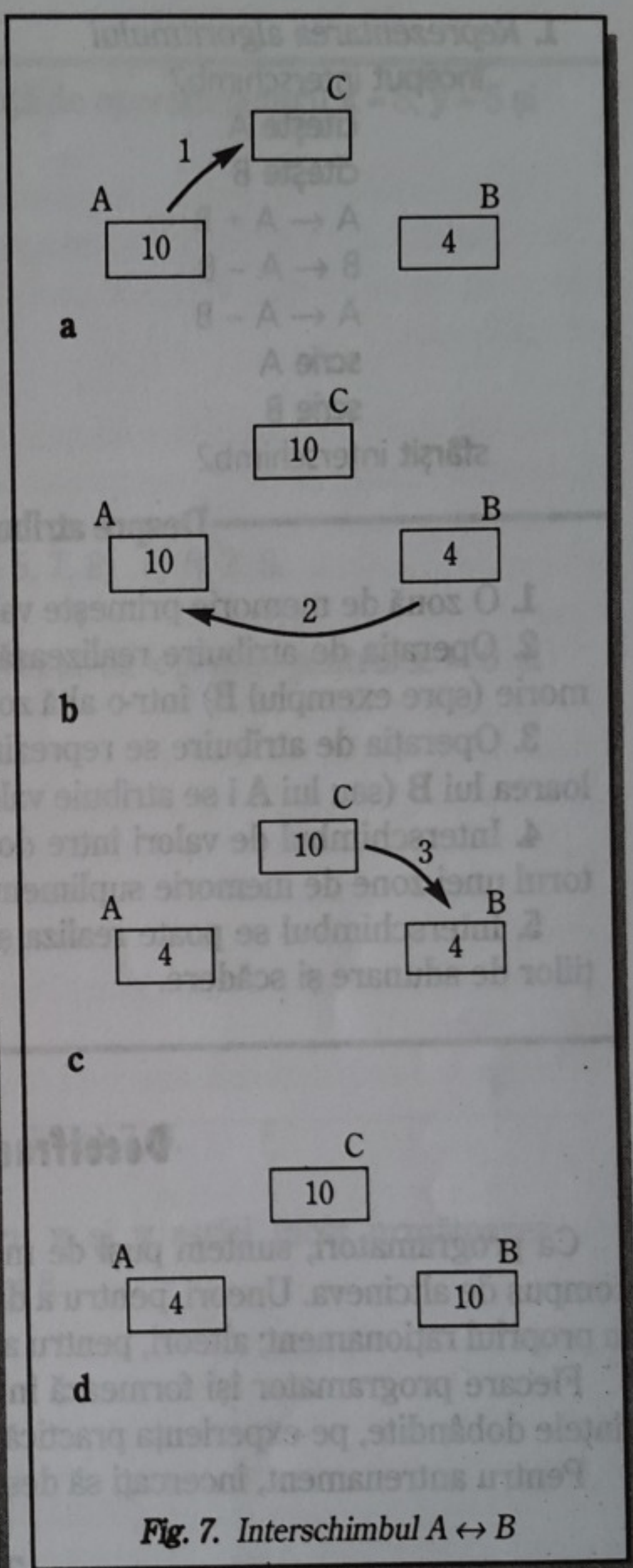
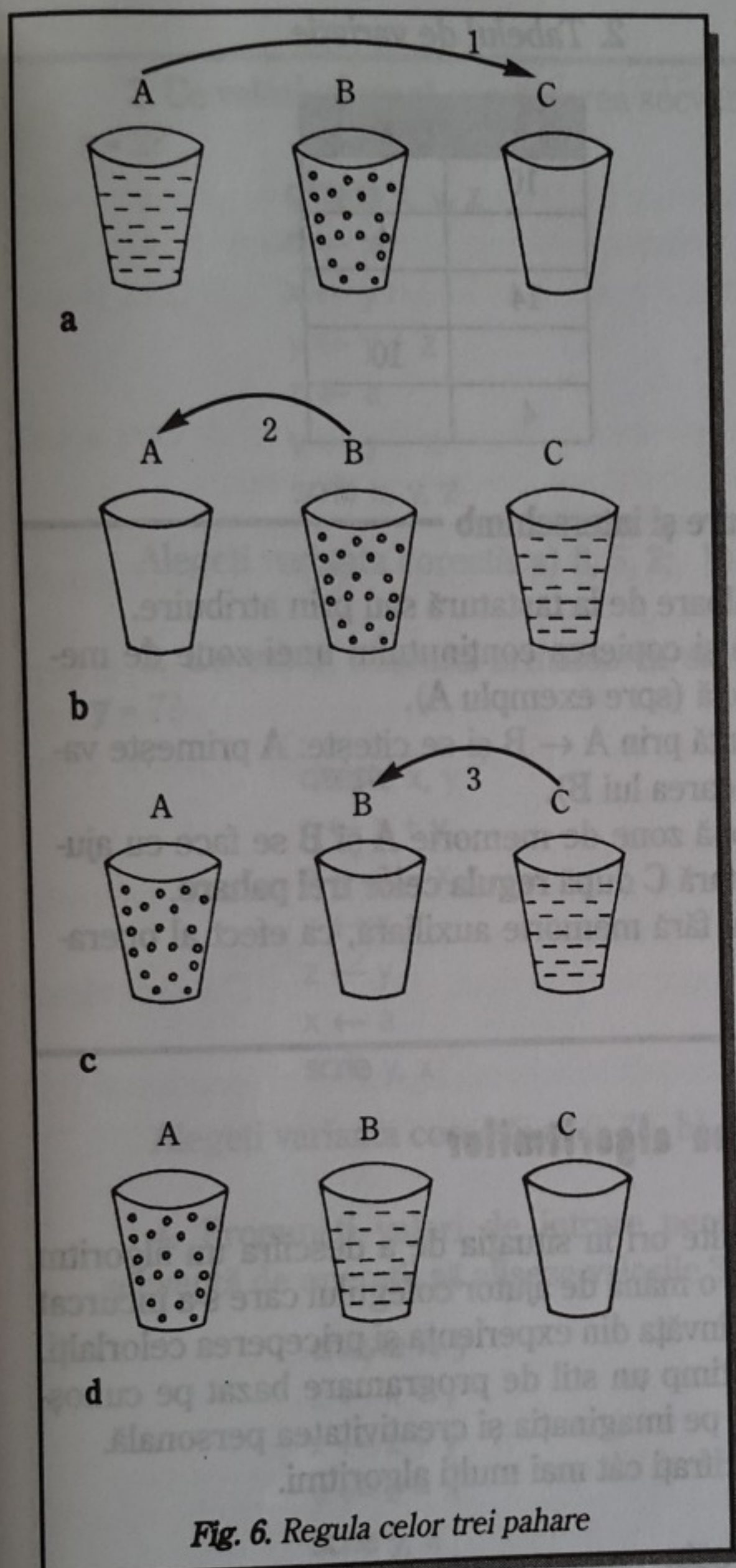
3. Tabelul de variație

Pentru verificarea corectitudinii folosim tabelul de variație.

Completați singuri spațiile libere.

Să căutăm și un alt raționament care să nu necesite memorie suplimentară.

Pentru valorile numerice ale celor două zone de memorie căutăm operații matematice care, aplicate succesiv, să aibă ca efect modificarea valorilor conform interschimbului cerut.



Inițial, **A** are valoarea 10, iar **B** valoarea 4. Dacă adunăm în **A** valoarea lui **B**, **A** devine 14.

Extragem din **A**, prin scădere, valoarea lui **B** și rezultatul îl păstrăm în **B**. **B** devine 10.

Nu ne rămâne decât să extragem încă o dată valoarea lui **B** din **A** și să păstrăm rezultatul în **A**. **A** devine 4.

1. Reprezentarea algoritmului

început interschimb2

citește A

citește B

$A \leftarrow A + B$

$B \leftarrow A - B$

$A \leftarrow A - B$

scrie A

scrie B

sfârșit interschimb2

2. Tabelul de variație

A	B
10	
	4
14	
	10
4	

Despre atribuire și interschimb

1. O zonă de memorie primește valoare de la tastatură sau prin atribuire.
2. Operația de atribuire realizează și copierea conținutului unei zone de memorie (spre exemplul **B**) într-o altă zonă (spre exemplu **A**).
3. Operația de atribuire se reprezintă prin $A \leftarrow B$ și se citește: **A** primește valoarea lui **B** (sau lui **A** i se atribuie valoarea lui **B**).
4. Interschimbul de valori între două zone de memorie **A** și **B** se face cu ajutorul unei zone de memorie suplimentară **C** după **regula celor trei pahare**.
5. Interschimbul se poate realiza și fără memorie auxiliară, ca efect al operațiilor de adunare și scădere.

Descifrarea algoritmilor

Ca programatori, suntem puși de multe ori în situația de a descifra un algoritm compus de altcineva. Uneori, pentru a da o mână de ajutor colegului care s-a încurcat în propriul raționament; alteori, pentru a învăța din experiența și priceperea celorlalți.

Fiecare programator își formează în timp un stil de programare bazat pe cunoștințele dobândite, pe experiența practică, pe imaginația și creativitatea personală.

Pentru antrenament, încercați să descifrați cât mai mulți algoritmi.

Teste

1. Ce valori afișează următoarea secvență de operații pentru $x = 3$ și $y = 4$?

citește x, y

$a \leftarrow x$

$y \leftarrow x$

$x \leftarrow a$

scrie x, y

Alegeți varianta corectă: a) 3, 4; b) 3, 3; c) 4, 3.

2. Ce valori afișează următoarea secvență de operații pentru $x = 8$, $y = 5$ și $z = 2$?

citește x, y, z

$a \leftarrow x$

$x \leftarrow y$

$y \leftarrow y + z$

$z \leftarrow a$

$y \leftarrow y - x$

scrie x, y, z

Alegeți varianta corectă: a) 8, 5, 2; b) 5, 7, 8; c) 5, 2, 8.

3. Ce valori afișează următoarea secvență de operații pentru $x = 6$ și $y = 7$?

citește x, y

$z \leftarrow x + y$

$a \leftarrow z - x$

$y \leftarrow x$

$z \leftarrow y$

$x \leftarrow a$

scrie y, x

Alegeți varianta corectă: a) 6, 7; b) 6, 6; c) 7, 6.

4. Propuneți valori de intrare pentru x și y astfel încât următoarea secvență de operații să afișeze valorile 3 și 8.

citește x, y

$z \leftarrow x + y$

$x \leftarrow z - y$

$y \leftarrow z - x$

scrie y, x

5. Propuneți valori de intrare pentru x , y și z astfel încât următoarea secvență de operații să afișeze valorile 3, 2, 1.

citește x, y, z

$a \leftarrow x$

$x \leftarrow z$

$z \leftarrow a$

$y \leftarrow y \div 3$

scrie x, y, z

9. Adevărat sau fals

Plec într-o tabără de vară. Îmi pregătesc un bagaj foarte ușor: câteva tricouri și pantaloni scurți. „Dacă plouă?” intervine bunica și adaugă pelerina, umbrela, haine și pantofi de schimb. La întoarcere, nu uit să spun îmbufnat: „am cărat o mulțime de lucruri inutile!”. „Și dacă ploua?” adaugă bunica.

Cu cât analizăm mai bine o situație necunoscută, descoperim factori de care aceasta depinde, cauze și efecte pe care să le folosim sau de care să ne ferim.

Ca programatori, nu strică să împrumutăm din înțelepciunea bunicii și să ținem seama de condițiile care influențează soluția unei probleme. Puține sunt problemele a căror rezolvare cu calculatorul să nu necesite verificarea unor condiții.

Banala împărțire a două numere **A** și **B** nu este întotdeauna o operație simplă pentru calculator. Ea devine chiar imposibilă dacă **B** are valoarea zero. Pus să rezolve o astfel de operație, calculatorul reacționează cu o decizie severă: întrerupe și abandonează programul. Din vina cui? A programatorului.

Verificarea condițiilor impune folosirea structurilor decizionale.

Structura decizională se compune dintr-un punct de control în care se trece condiția și secvențele de operații necesare pentru fiecare caz al condiției.

Condiția se formulează astfel încât să accepte doar două răspunsuri: **DA (adevărat)** și **NU (fals)**.

În figura 8 este descrisă structura decizională în schemă logică (a) și pseudocod (b).

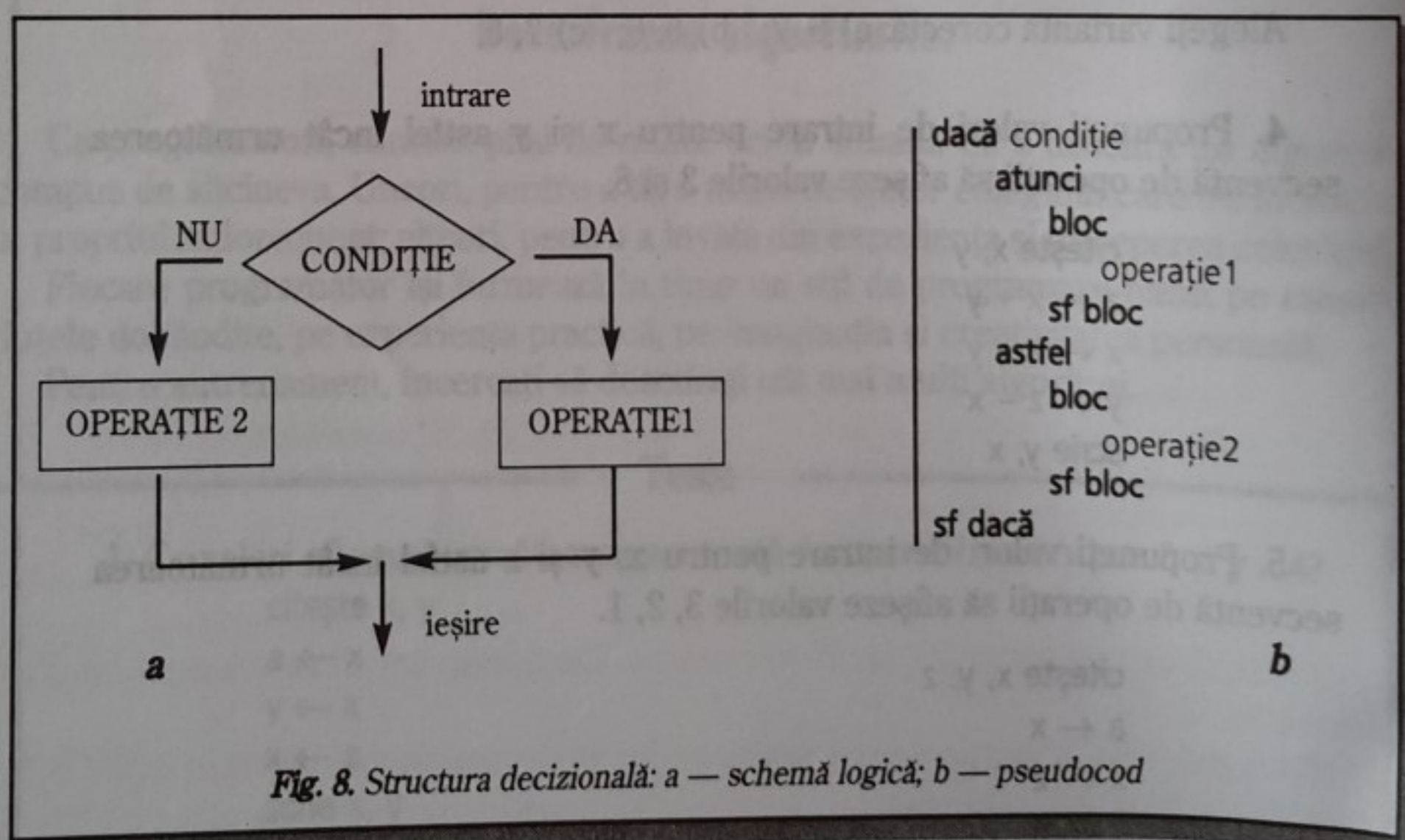


Fig. 8. Structura decizională: a — schemă logică; b — pseudocod

Reprezentarea în schemă logică pune foarte bine în evidență punctul de intrare în blocul de decizie și punctul de ieșire (închiderea celor două ramuri).

Problema împărțirii lui A la B rezolvată secvențial (figura 9a) nu are o soluție sigură întrucât algoritmul este greșit; nu respectă proprietatea de a fi general.

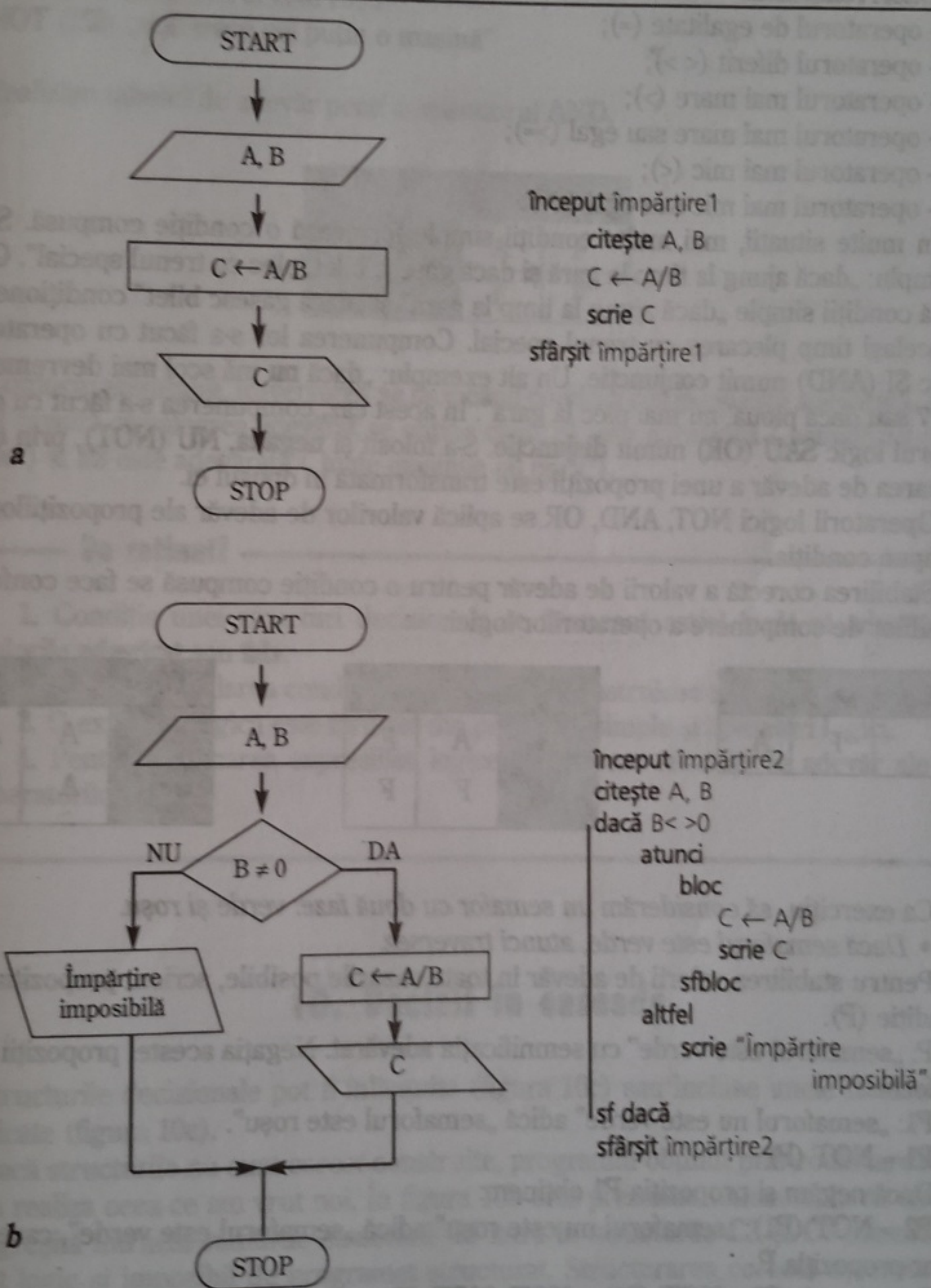


Fig. 9. Împărțirea lui A la B : a — algoritm secvențial (greșit); b) algoritm decizional

Rezolvarea trebuie modificată astfel încât să cuprindă și situația $B = 0$. Algoritmul decizional din figura 9b rezolvă toate cazurile problemei.

Pentru formularea condițiilor se folosesc enunțuri simple, numite **propoziții**, și **operatori relaționali**:

- operatorul de egalitate ($=$);
- operatorul diferit ($<>$);
- operatorul mai mare ($>$);
- operatorul mai mare sau egal ($>=$);
- operatorul mai mic ($<$);
- operatorul mai mic sau egal ($<=$).

În multe situații, mai multe condiții simple formează o condiție compusă. Spre exemplu: „dacă ajung la timp la gară și dacă găsesc bilet, plec cu trenul special”. Cele două condiții simple „dacă ajung la timp la gară” și „dacă găsesc bilet” condiționează în același timp plecarea cu trenul special. Compunerea lor s-a făcut cu operatorul logic **ȘI (AND)** numit **conjuncție**. Un alt exemplu: „dacă nu mă scol mai devreme de ora 7 sau dacă plouă, nu mai plec la gară”. În acest caz, compunerea s-a făcut cu operatorul logic **SAU (OR)** numit **disjuncție**. S-a folosit și **negația, NU (NOT)**, prin care valoarea de adevăr a unei propoziții este transformată în opusul ei.

Operatorii logici NOT, AND, OR se aplică valorilor de adevăr ale propozițiilor ce compun condiția.

Stabilirea corectă a valorii de adevăr pentru o condiție compusă se face conform regulilor de compunere a operatorilor logici:

NOT	A	F
	F	A

AND	A	F
A	A	F
F	F	F

OR	A	F
A	A	A
F	A	F

Ca exercițiu, să considerăm un semafor cu două faze: verde și roșu.

- Dacă semaforul este verde, atunci traversez.

Pentru stabilirea valorii de adevăr în toate cazurile posibile, scriem propoziția din condiție (**P**).

P: „semaforul este verde” cu semnificația **adevărat**. Negația acestei propoziții este de forma:

P1: „semaforul nu este verde” adică „semaforul este roșu”.

P1 = NOT (**P**)

Dacă negăm și propoziția **P1** obținem:

P2 = NOT (**P1**): „semaforul nu este roșu” adică „semaforul este verde”, care este chiar propoziția **P**.

Putem scrie **P2** = NOT (**P1**) = NOT (NOT (**P**)) = **P**.

- Dacă semaforul este verde și dacă toate mașinile au oprit, atunci traversez.

În acest caz, avem de evaluat o condiție compusă.

Scriem propozițiile simple din condiție:

P1: „semaforul este verde” cu semnificația **adevărat**.

P2: „toate mașinile au oprit” cu semnificația **adevărat**.

Rezultă:

NOT (P1): „semaforul este roșu”

NOT (P2): „mai trece cel puțin o mașină”

Realizăm tabelul de adevăr pentru operatorul **AND**.

P1 AND P2	A	F
A	A	F
F	F	F

Conform enunțului, traversarea se face când condiția compusă este adevărată. Din tabel rezultă că valoarea **adevărat** se obține când **P1** este adevărată („semaforul este verde”) și **P2** este adevărată („toate mașinile au oprit”).

De reținut!

1. Condiția unei structuri decizionale se formează astfel încât să admită valorile **adevărat** sau **fals**.
2. Pentru formularea condițiilor compuse se construiesc expresii logice.
3. O expresie logică este formată din propoziții simple și operatori logici.
4. Pentru verificarea expresiilor logice se folosesc tabelele de adevăr ale operatorilor logici.

10. Decizii în cascadă

Structurile decizionale pot fi înlănțuite (figura 10c) sau incluse unele în altele – imbricate (figura 10c).

Dacă structurile nu sunt corect construite, programul obținut prin codificarea lor nu va realiza ceea ce am vrut noi. În figura 10b este prezentat un exemplu de structură greșită întrucât ramurile structurii **C3** intră în structurile **C2** și **C1** „amestec” greșit logic și imposibil de programat structurat. Structurarea corectă corespunde reprezentării din figura 10c.

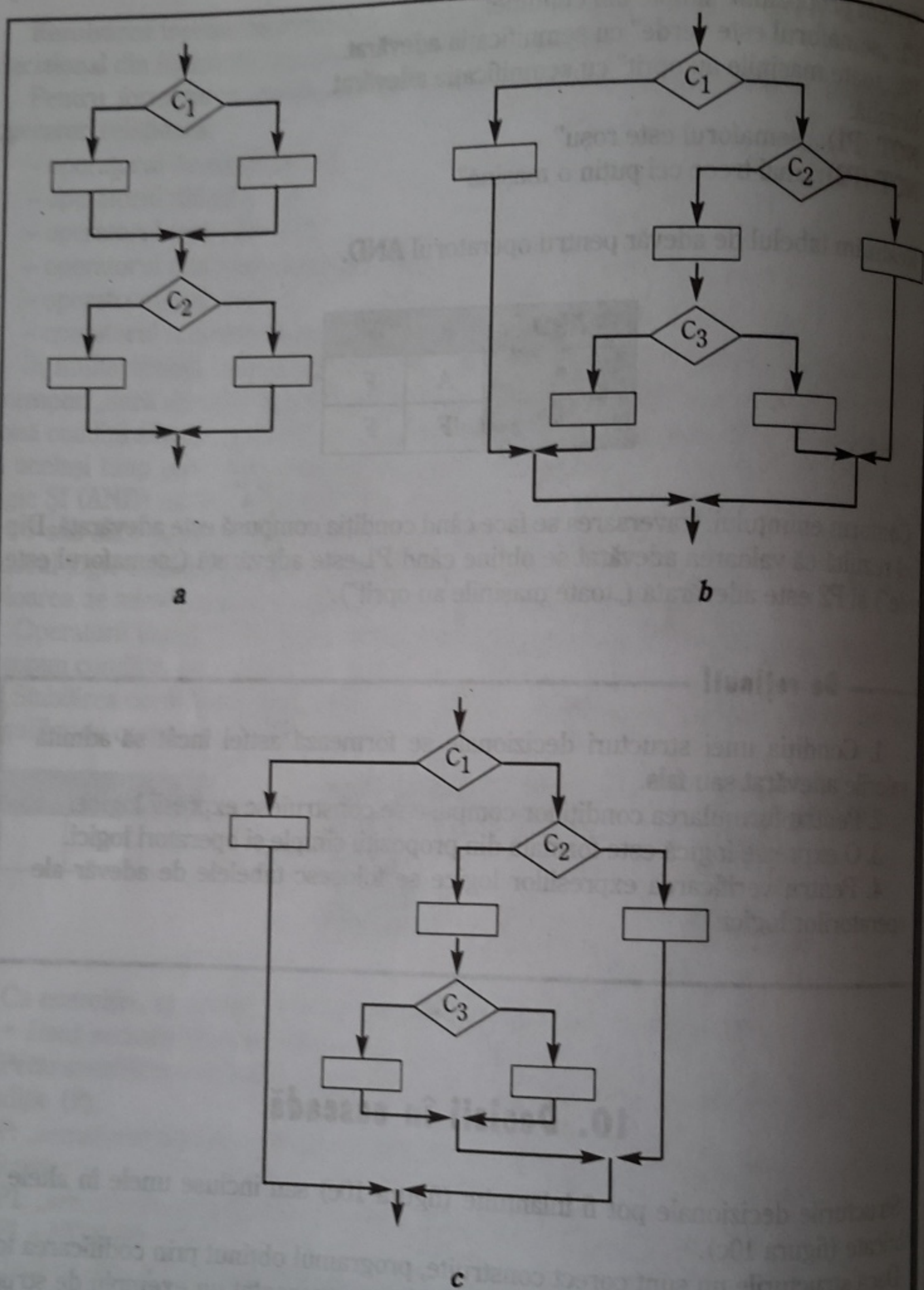


Fig. 10. Structuri decizionale în cascadă: a — în lanțuite;
b — imbricante (greșit); c — imbricate (corect)

Vom exemplifica folosirea structurilor decizionale înlănțuite sau incluse pentru următorul exemplu:

Trei copii participă la un concurs. Cunoscând punctajul fiecărui copil la sfârșitul concursului, să se determine punctajul cel mai mare pentru stabilirea câștigătorului.

Fie A , B , C punctajul celor trei copii și câteva exemple concrete (vezi tabelul 4).

Tabelul 4

A	B	C	Punctajul maxim
10	5	25	25
100	50	85	100
35	90	89	90
100	100	90	100
16	32	32	32

Rezolvarea problemei se reduce la compararea punctajelor A , B , C pentru determinarea valorii maxime.

Ca soluție, propunem doi algoritmi (figura 11). Reprezentarea prin înlănțuirea deciziilor folosește mai puține puncte de control și, ca urmare, descrie mai puține ramificații. Codificarea acestui algoritm va genera un program mai scurt.

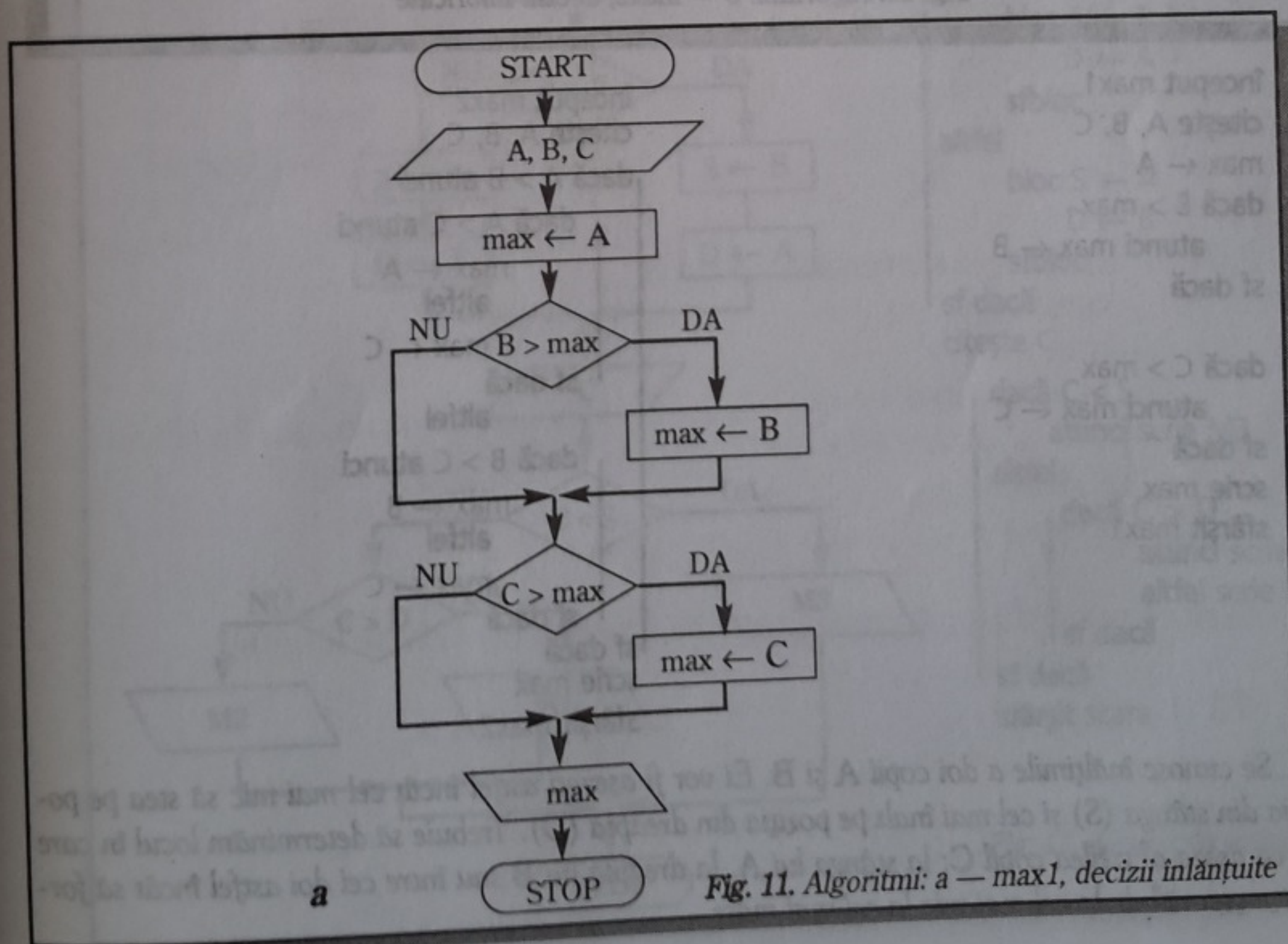


Fig. 11. Algoritmi: a — max1, decizii înlănțuite

Reprezentarea prin structuri imbricate necesită mai multă atenție la închidere corectă a acestora.

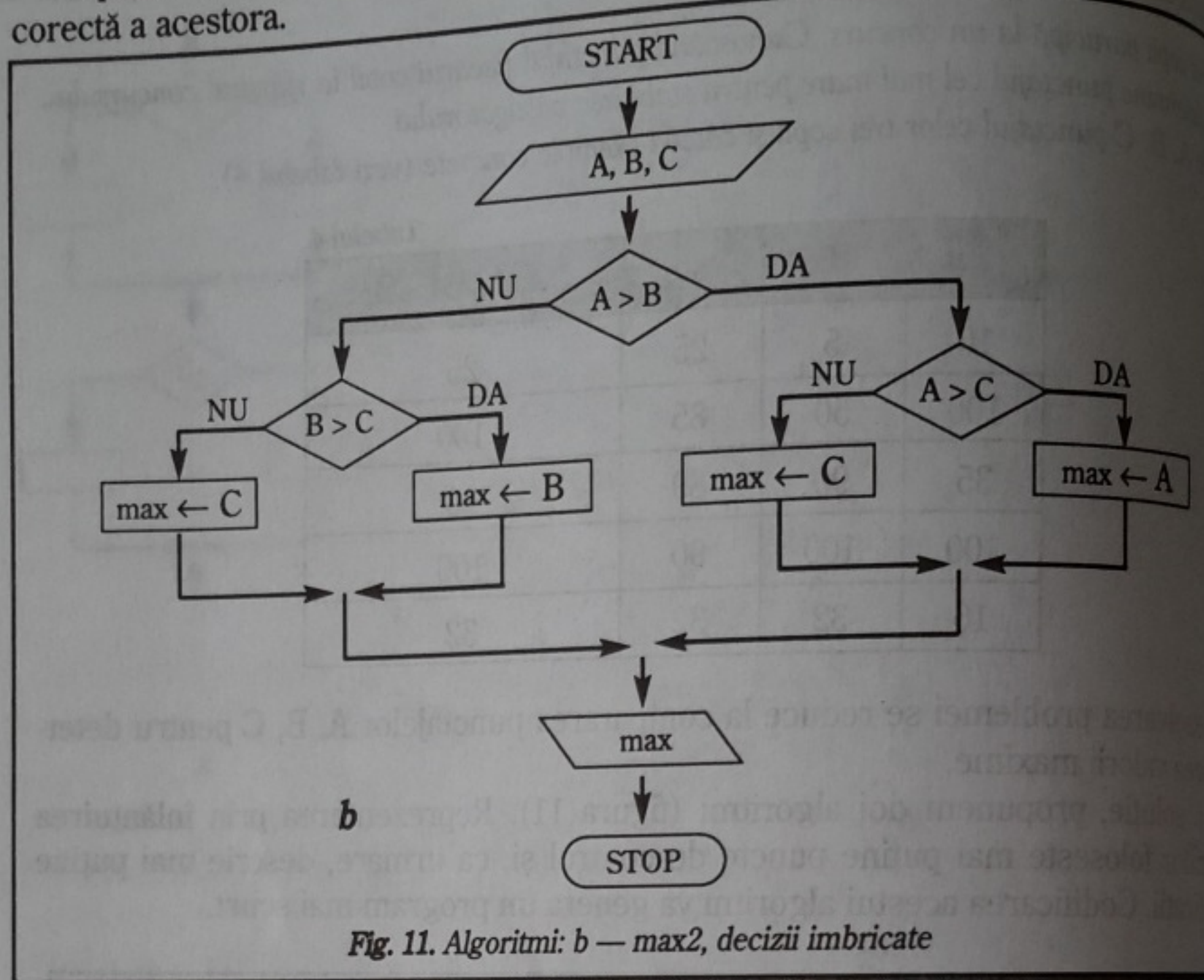


Fig. 11. Algoritmi: b — max2, decizii imbricate

```

început max1
citește A, B, C
max ← A
dacă B > max
    atunci max ← B
sf dacă

dacă C > max
    atunci max ← C
sf dacă
scrie max
sfârșit max1
  
```

```

început max2
citește A, B, C
dacă A > B atunci
    dacă A > C atunci
        max ← A
    altfel
        max ← C
    sf dacă
    altfel
        dacă B > C atunci
            max ← B
        altfel
            max ← C
        sf dacă
    sf dacă
scrie max
sfârșit max2
  
```

Se cunosc înălțimile a doi copii A și B. Ei vor fi așezați astfel încât cel mai mic să stea pe poziția din stânga (S) și cel mai înalt pe poziția din dreapta (D). Trebuie să determinăm locul în care se va așeza al treilea copil C: la stânga lui A, la dreapta lui B sau între cei doi astfel încât să formeze „scară” de la cel mai mic la cel mai mare.

Pentru început se compară înălțimile **A** cu **B** și se determină cine va sta în stânga și cine în dreapta. Copilul **C** se compară cu **S**; dacă e mai mărunț va sta în stânga sa, altfel se compară cu **D**. Dacă e mai înalt decât **D**, va sta la dreapta acestuia, altfel se va așeza între **S** și **D**.

Exemplu numeric:

Tabelul 5

A	B	S	D	C	Poziția lui C	Mesaj
149	145	145	149	150	la dreapta lui D	M1
145	160	145	160	150	între S și D	M2
165	143	143	165	140	la stânga lui S	M3

Algoritmul problemei este prezentat în figura 12. În reprezentarea acestuia s-a folosit atât înălțuirea cât și includerea structurilor minim necesare.

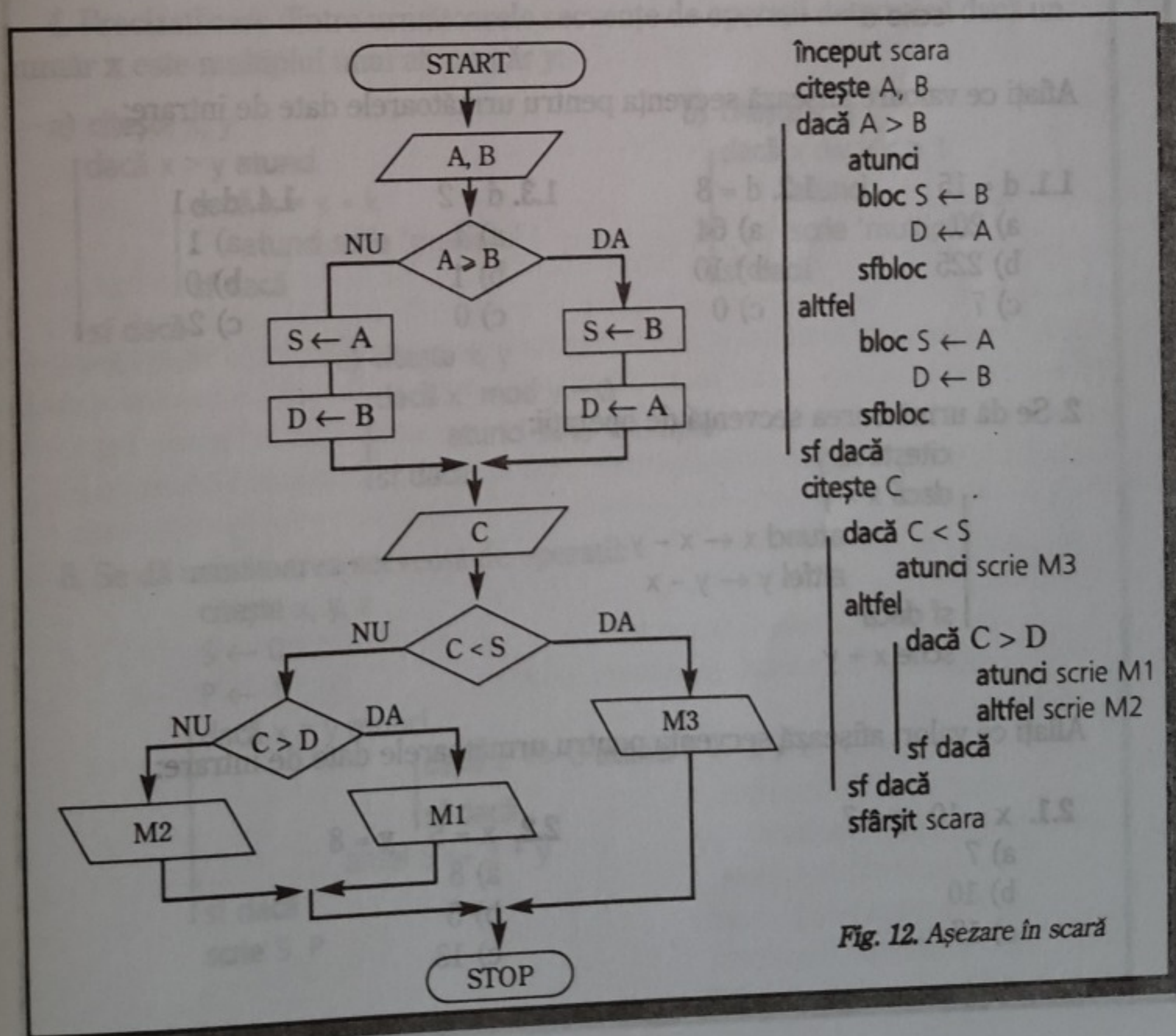


Fig. 12. Așezare în scară

Scurt și simplu

Un algoritm scurt și simplu va genera un program ușor de scris și de verificat.

Teste

1. Se dă următoarea secvență de operații:

```
citește d
dacă d > 10
    atunci d ← d * d
    altfel
        dacă d < 5
            atunci
                d ← d + d
            altfel
                d ← d mod 2
        sf dacă
    sf dacă
scrie d
```

Aflați ce valoare afișează secvența pentru următoarele date de intrare:

1.1. d = 15

- a) 20
- b) 225
- c) 7

1.2. d = 8

- a) 64
- b) 10
- c) 0

1.3. d = 2

- a) 4
- b) 1
- c) 0

1.4. d = 1

- a) 1
- b) 0
- c) 2

2. Se dă următoarea secvență de operații:

```
citește x, y
dacă x > y
    atunci x ← x - y
    altfel y ← y - x
sf dacă
scrie x + y
```

Aflați ce valori afișează secvența pentru următoarele date de intrare:

2.1. x = 10 y = 7

- a) 7
- b) 10
- c) 17

2.2. x = 5 y = 8

- a) 8
- b) 5
- c) 13

3. Se dă următoarea secvență de operații:

citește x, y

$z \leftarrow x$

dacă $x = y$ atunci $z \leftarrow z + z$
altfel

dacă $x > y$ atunci $z \leftarrow z - y$

altfel $z \leftarrow y + z - x$

sf dacă

sf dacă

scrie z

3.1. Dați exemple de valori pentru care secvența dată afișează suma celor două numere x și y .

3.2. Dați exemple de valori pentru care secvența dată afișează cea mai mare dintre valorile x și y .

4. Precizați care dintre următoarele secvențe de operații determină dacă un număr x este multiplul unui alt număr y :

a) citește x, y

dacă $x > y$ atunci

dacă $x = y * k$

atunci scrie 'multiplu'

sf dacă

sf dacă

b) citește x, y

dacă $x \text{ div } y \neq 1$

atunci

scrie 'multiplu'

sf dacă

c) citește x, y

dacă $x \bmod y = 0$

atunci scrie 'multiplu'

sf dacă

5. Se dă următoarea secvență de operații:

citește x, y, z

$S \leftarrow 0$

$P \leftarrow 1$

dacă $x > y$ atunci

dacă $z \neq 0$ atunci $P \leftarrow x * z$

sf dacă

altfel $S \leftarrow z + y$

sf dacă

scrie S, P

5.1. Dați exemple de valori x, y, z pentru care această secvență afișează valorile 0 și 1.

5.2. Dați exemple de valori x, y, z pentru care această secvență afișează valorile 0 și 0.

5.3. Dați exemple de valori pentru care această secvență să afișeze valoarea lui y și 1.

6. Precizați ce realizează următoarea secvență de operații:

citește x, y

dacă $(x < 0)$ și $(y < 0)$

atunci

$E \leftarrow 1/x + 1/y$

sf dacă

scrie E

a) calculează și afișează valoarea unei expresii oarecare E ;

b) calculează și afișează valoarea expresiei $E = (x + y)/x \times y$;

c) verifică numitorii nenuli a două fracții și calculează suma lor.

11. Din nou la ZooCarnaval

Regizorul teatrului de păpuși în care se pregătește spectacolul **ZooCarnaval** este foarte distras. Nu mai știe câte animale și câte păsări participă la carnaval. Nici șeful costumierilor nu-i poate spune exact, dar îl ajută dându-i numărul de pălării și numărul de pantofi și mănuși pe care le-a confecționat. Regizorul nu se pricepe la matematică și cere disperat ajutorul nostru.

1. Analiza problemei

- Ce știm:

- numărul de pălării (capete);
- numărul de pantofi și mănuși (picioare).

- Ce se cere:

- numărul de păsări;
- numărul de animale.

Notăm datele problemei:

- c – numărul de capete;
- p – numărul de picioare;
- x – numărul de păsări;
- y – numărul de animale.

2. Raționamentul problemei

- fiecare pasăre are un cap și două picioare;
- fiecare animal are un cap și patru picioare.

Rezultă: $c = x + y$

$$p = 2 * x + 4 * y$$

din relația (1) rezultă $x = c - y$

relația (2) devine $p = 2 * (c - y) + 4 * y = 2 * c + 2 * y$

din relația (4) se obține $y = (p - 2 * c) / 2$

înlocuind pe y în relația (3) se obține $x = c - (p - 2 * c) / 2$

$$x = \frac{2 * c - p + 2 * c}{2} = \frac{4 * c - p}{2}$$

- Formulăm exemple concrete (vezi tabelul 5).

Tabelul 5

c	p	x	y	Observații
5	10	5	0	
8	20	6	2	
3	5			imposibil
8	6			imposibil
				imposibil

Completați singuri tabelul cu alte valori pentru care problema este imposibilă.

Din exemplele concrete deducem că datele de intrare (c și p) trebuie să corespundă semnificațiilor din problemă (nu orice valori au sens sau se potrivesc unei situații reale).

Astfel:

- numărul de picioare trebuie să fie multiplu de doi (să fie număr par);
- numărul de picioare trebuie să fie „suficient” (cel puțin $2 * c$), dar nu „prea mare” (să nu depășească $4 * c$).

3. Condiții și relații importante

p — număr par

$$2 * c \leq p \leq 4 * c$$

$$x = \frac{4 * c - p}{2}$$

$$y = \frac{p - 2 * c}{2}$$

4. Reprezentarea algoritmului

```
început
citește p, c
dacă  $p \bmod 2 = 1$ 
    atunci
        scrie "problema nu are soluție"
    altfel
        dacă  $(2 * c > p)$  SAU  $(4 * c < p)$ 
            atunci
                scrie "problema nu are soluție"
            altfel
                bloc
                     $x \leftarrow (4 * c - p) \div 2$ 
                     $y \leftarrow (p - 2 * c) \div 2$ 
                    scrie x, y
                sf bloc
        sf dacă
    sf dacă
sfârșit
```

12. Selecția

Andrei este în clasa a V-a și își dorește un program care să afișeze pe ecranul calculatorului un mesaj diferit de la o zi la alta a săptămânii (vezi tabelul 6).

Tabelul 6

Ziua	Mesajul	Semnificația
luni	M1	îți doresc o săptămână ușoară
marți	M2	ziua cea mai lungă
miercuri	M3	va fi o zi obișnuită
joi	M4	vei avea multe surprize frumoase
vineri	M5	mai e puțin până sâmbătă
sâmbătă	M6	distrează-te
duminică	M7	nu uita să te joci

Andrei și-a făcut și un plan de rezolvare: „Citesc ziua de la tastatură. Ca să meargă mai repede, în loc de luni, marți, ..., voi introduce 1, 2, ... Pare genul de program care poate fi scris direct fără schemă logică sau pseudocod.”

Pentru că este începător, Andrei a hotărât să mai exerseze puțin reprezentările. Deși problema e clară, a căutat o cale cât mai simplă prin care să-i explice calculatorului ce mesaj să afișeze în funcție de zi. Nu a găsit altă soluție decât să compare ziua introdusă de la tastatură cu fiecare valoare în parte de la 1 la 7 și să scrie mesajul potrivit.

Concluzia: „multe condiții. Dacă aleg varianta cu structuri incluse, trebuie să fiu atent să le închid corect.” Andrei a ales reprezentarea prin structuri înlanțuite (figura 13). Algoritmul este corect, dar folosește 7 structuri.

Algoritmul poate fi reprezentat mai simplu prin **structura de tip selecție**.

Structura de tip **selecție** este o structură cu ramificații dirijate de valoarea unui selector plasat în punctul de control. Fiecare ramură corespunde unei valori a selectorului; această valoare se trece ca o etichetă în fața blocului de operații. Dacă valoarea selectorului nu apare pe nici una dintre etichete, execuția este dirijată pe o ramură specială (figura 14).

Structura de **selecție** este ușor de codificat și folosirea ei reduce foarte mult dimensiunea programelor.

În problema lui Andrei, selectorul este chiar ziua. Valorile corecte sunt 1, 2 până la 7; pentru orice altă valoare, se poate afișa un mesaj special de eroare. În figura 15 este reprezentat algoritmul complet al problemei. Faceți singuri comparația cu reprezentarea din figura 13.

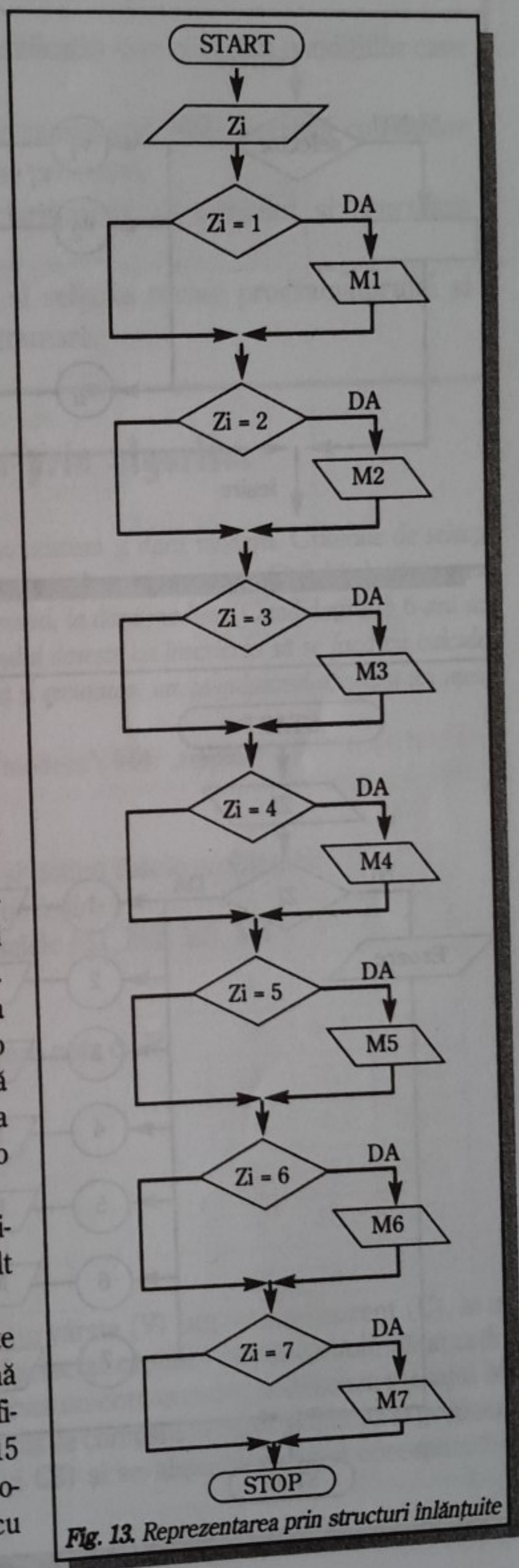
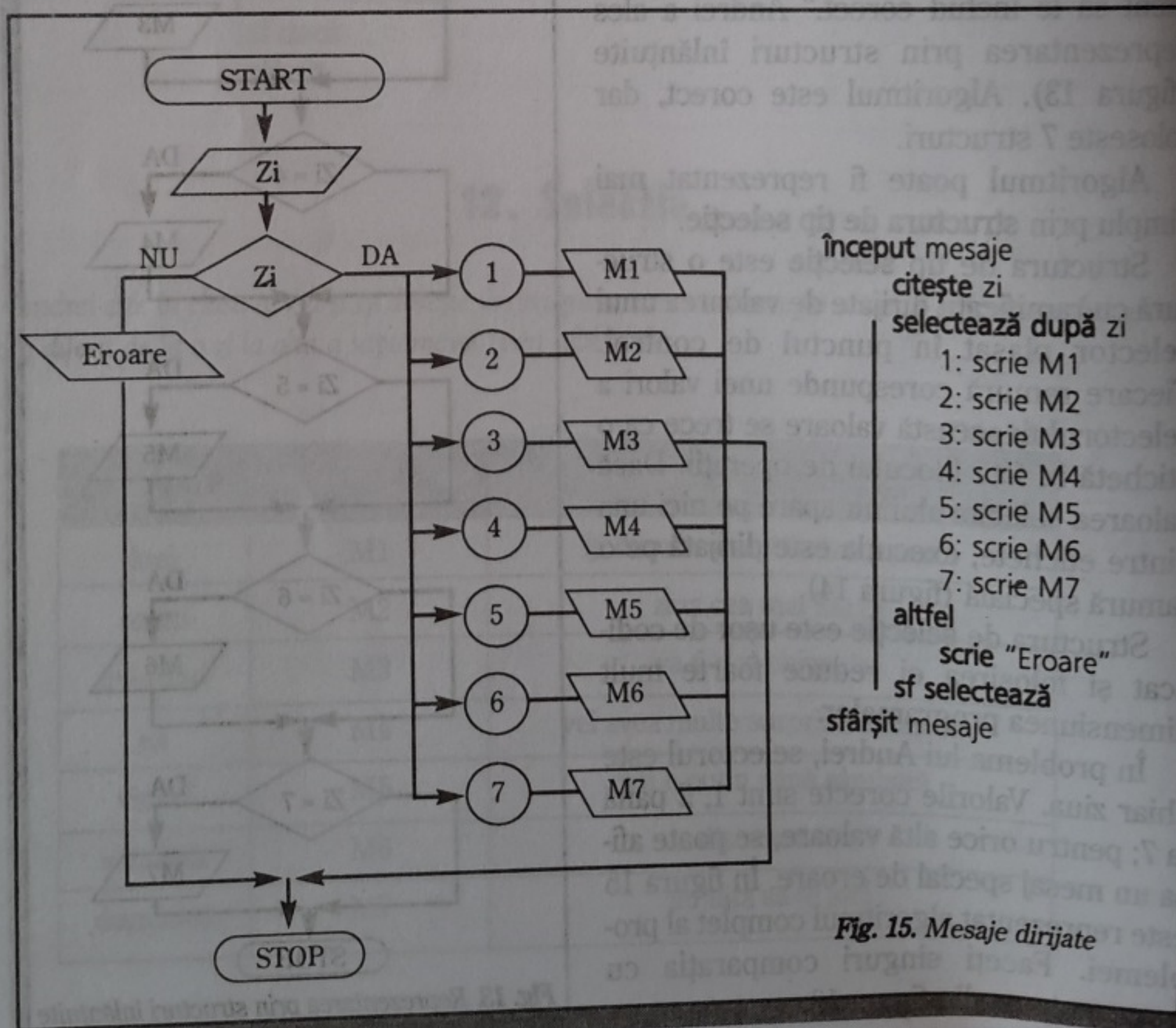
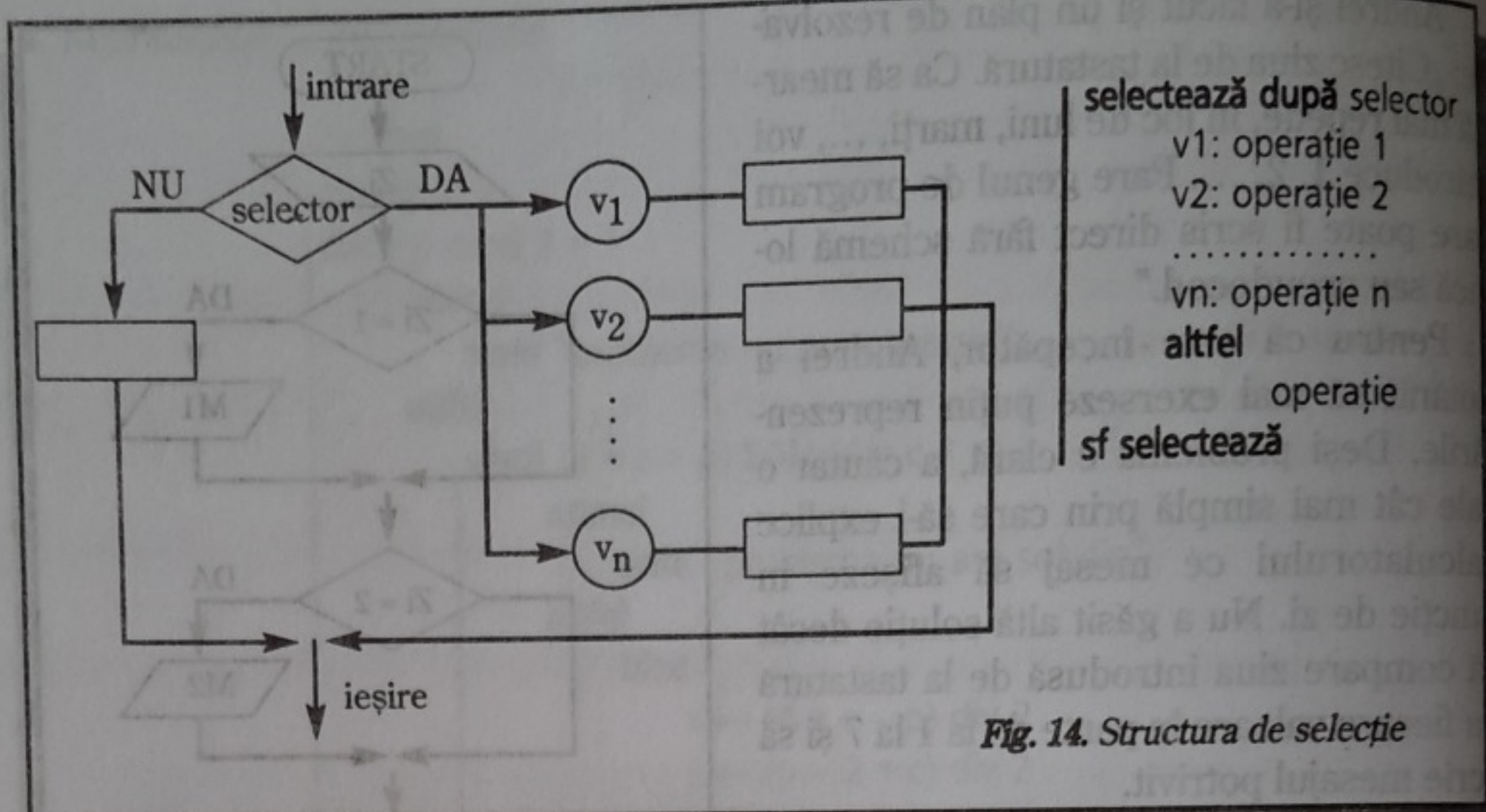


Fig. 13. Reprezentarea prin structuri înlanțuite



Când luăm decizii de tip selecție

1. Decizia de tip DA sau NU (cu două ramificații) este specifică condițiilor care pot lua doar două valori: **adevărat** sau **fals**.
2. Decizia de tip **selecție** (cu mai multe ramificații) este specifică condițiilor care pot lua valori într-o mulțime fixă și bine precizată.
3. Folosirea **selecției** contribuie la clarificarea algoritmului și scurtează pseudocodul sau programul.
4. Alegerea între deciziile în cascadă și **selecție** revine programatorului și depinde de experiența și stilul său de programare.

13. Pas cu pas prin algoritm

Un club sportiv selectează copii pentru natație, atletism și dans modern. Criteriile de selecție sunt vârsta, înălțimea și greutatea. Copiii sub 1,50 m sunt înscriși la natație. Ceilalți, dacă au greutatea mai mare de 45 kg sunt înscriși la atletism; restul, la dans modern. Candidații sub 6 ani sau mai mari de 10 ani sunt respinși. Directorul clubului dorește ca înscrierea să se facă cu calculatorul: candidatul introduce anul nașterii, înălțimea și greutatea, iar calculatorul afișează un mesaj corespunzător:

M1: „natație”; M2: „atletism”; M3: „dans modern”; M4: „respins”.

1. Analiza problemei

După lectura atentă a textului, stabilim și notăm datele problemei:

- Date de intrare
 - anul nașterii **A**
 - înălțimea **H**
 - greutatea **G**
- Date de ieșire
 - mesajele **M1, M2, M3, M4**

În problemă apare și vârsta pe care o vom nota cu **V**.

- Condiții și relații importante

$$V = C - A$$

$$C1: (V \geq 6) \text{ și } (V \leq 10)$$

$$C2: H < 1,50$$

$$C3: G < 45$$

2. Raționamentul problemei

Cunoscând anul nașterii (**A**) determinăm vârsta (**V**) față de anul curent (**C**), în cazul nostru 2004. Verificăm dacă poate fi selectat copilul: vârsta trebuie să aparțină intervalului [6, 10] – condiția **C1**. Dacă vârsta nu corespunde, se afișează mesajul **M4** și programul se termină. Pentru copilul admis se compară pe rând înălțimea și greutatea cu limitele din problemă (condițiile **C2** și **C3**) și se afișează mesajul corespunzător înscrierii făcute.

3. Reprezentarea algoritmului

```

început club-sportiv
  C ← 2004 // anul curent poate fi extras din data calculatorului
  citește A, H, G
  V ← C - A
  dacă (V ≥ 6) și (V ≤ 10)
  |
  |   atunci
  |   |   dacă H < 1.50
  |   |   |   atunci scrie M1
  |   |   |   altfel
  |   |   |   |   dacă G < 45
  |   |   |   |   |   atunci scrie M3
  |   |   |   |   |   altfel scrie M2
  |   |   |   |   sf dacă
  |   |   |   sf dacă
  |   |   altfel scrie M4
  |   sf dacă
sfârșit club-sportiv
  
```

4. Cum funcționează algoritmul?

Pentru a urmări mecanismul algoritmului îl vom parcurge **pas cu pas**, completând în tabelul de variație (tabelul 7) valorile corespunzătoare zonelor de memorie și condițiilor.

Tabelul 7

Zone de memorie				Condiții			Mesaj
A	H	G	V	C1	C2	C3	
1994	1,45	45	10	A	A	—	M1
1996	1,55	43	8	A	F	A	M3
1987	1,57	55	17	F	—	—	M4
1995	1,58	50	9	A	F	F	M2
1990	1,51	50					
1996	1,53	49					
1999	1,10	35					

Completați singuri spațiile libere din tabel.

Din tabelul de variație rezultă clar modul de funcționare a algoritmului: la fiecare execuție, datele de intrare determină valoarea logică a condițiilor în funcție de care se execută operația corespunzătoare. Nu există mesaje (date de ieșire) în contradicție cu cerințele problemei.

De reținut!

1. Verificarea corectitudinii unui algoritm se face **pas cu pas** cu ajutorul tabelului de variație.
2. Datele de intrare se aleg astfel încât fiecare condiție să treacă prin cele două valori logice **adevărat** sau **fals**.

Teste

1. Se dă următoarea secvență de operații:

```
citește x
dacă x mod 2 = 0 atunci d ← 2
    altfel
        dacă x mod 3 = 0
            atunci d ← 3
            altfel
                dacă x mod 5 = 0
                    atunci d ← 5
                sf dacă
            sf dacă
        sf dacă
    sf dacă
scrie d
```

Precizați ce realizează această secvență:

- a) verifică dacă un număr **x** este multiplu de 2, 3 și 5;
- b) verifică dacă un număr **x** este par sau multiplu de 3;
- c) verifică dacă un număr **x** are ca divizor pe 2, pe 3 sau pe 5;
- d) afișează primul dintre divizorii 2, 3 sau 5 ai numărului **x**.

2. Se dă următoarea secvență de instrucțiuni:

```
citește a, b
S ← a + b
P ← a * b
dacă P = 0 atunci
    dacă a = 0 atunci S ← S - a
    altfel S ← S - b
    sf dacă
sf dacă
scrie S, P
```

Stabiliți pentru ce valori de intrare, **P** și **S** au următoarele valori:

- a) $P = 0$ și $S = a$;
- b) $P = 0$ și $S = b$;
- c) $P = 0$ și $S = 0$.

3. Precizați care dintre următoarele secvențe de operații verifică dacă x este cea mai mare valoare dintre valorile x , y și z :

a) citește x, y, z
 dacă $x > y$ atunci
 dacă $y > z$ atunci
 scrie 'x este maxim'
 sf dacă
 sf dacă

b) citește x, y, z
 dacă $y < x$ atunci
 dacă $z < x$ atunci
 scrie 'x este maxim'
 sf dacă
 sf dacă

c) citește x, y, z
 dacă $x > y$ atunci
 dacă $z < x$
 atunci
 scrie 'x este maxim'
 sf dacă
 sf dacă

d) citește x, y, z
 dacă $(x > y)$ sau $(x > z)$
 atunci
 scrie 'x este maxim'
 sf dacă

4. Precizați care dintre următoarele secvențe calculează suma elementelor pare citite:

a) citește x, y, z
 dacă $(x \bmod 2 = 0)$ și $(y \bmod 2 = 0)$ și $(z \bmod 2 = 0)$
 atunci $s \leftarrow x + y + z$
 sf dacă

b) citește x, y, z
 dacă $x \bmod 2 = 0$ atunci $s \leftarrow s + x$ sf dacă
 dacă $y \bmod 2 = 0$ atunci $s \leftarrow s + y$ sf dacă
 dacă $z \bmod 2 = 0$ atunci $s \leftarrow s + z$ sf dacă

c) citește x, y, z
 $s \leftarrow x + y + z$
 dacă $s \bmod 2 = 0$ atunci $s \leftarrow x + y + z$
 sf dacă

14. Repetiția condiționată

Algoritmii care se reprezintă prin structuri liniare sau cu ramificații (decizionale sau de selecție) au proprietatea că fiecare dintre operații se execută cel mult o dată.

Execuția repetată a unei secvențe de operații este posibilă doar în cadrul structurilor repetitive.

După modul în care este controlată repetiția, există două categorii de structuri repetitive: condiționate și cu număr cunoscut de repetiții.

În cadrul structurilor repetitive condiționate, repetarea unei operații se face în funcție de îndeplinirea unei condiții, fără să cunoaștem de la început numărul exact de repetiții („repet citirea poeziei până o știu pe de rost”). Când se cunoaște de la început numărul de repetiții („repet calculul mediei generale de 25 de ori, pentru fiecare elev al clasei”) nu mai este necesară o altă condiție de control.

Repetă până când

Repetarea secvenței de operații până la îndeplinirea unei condiții se face prin structura **repetă până când**, cu reprezentarea din figura 16a.

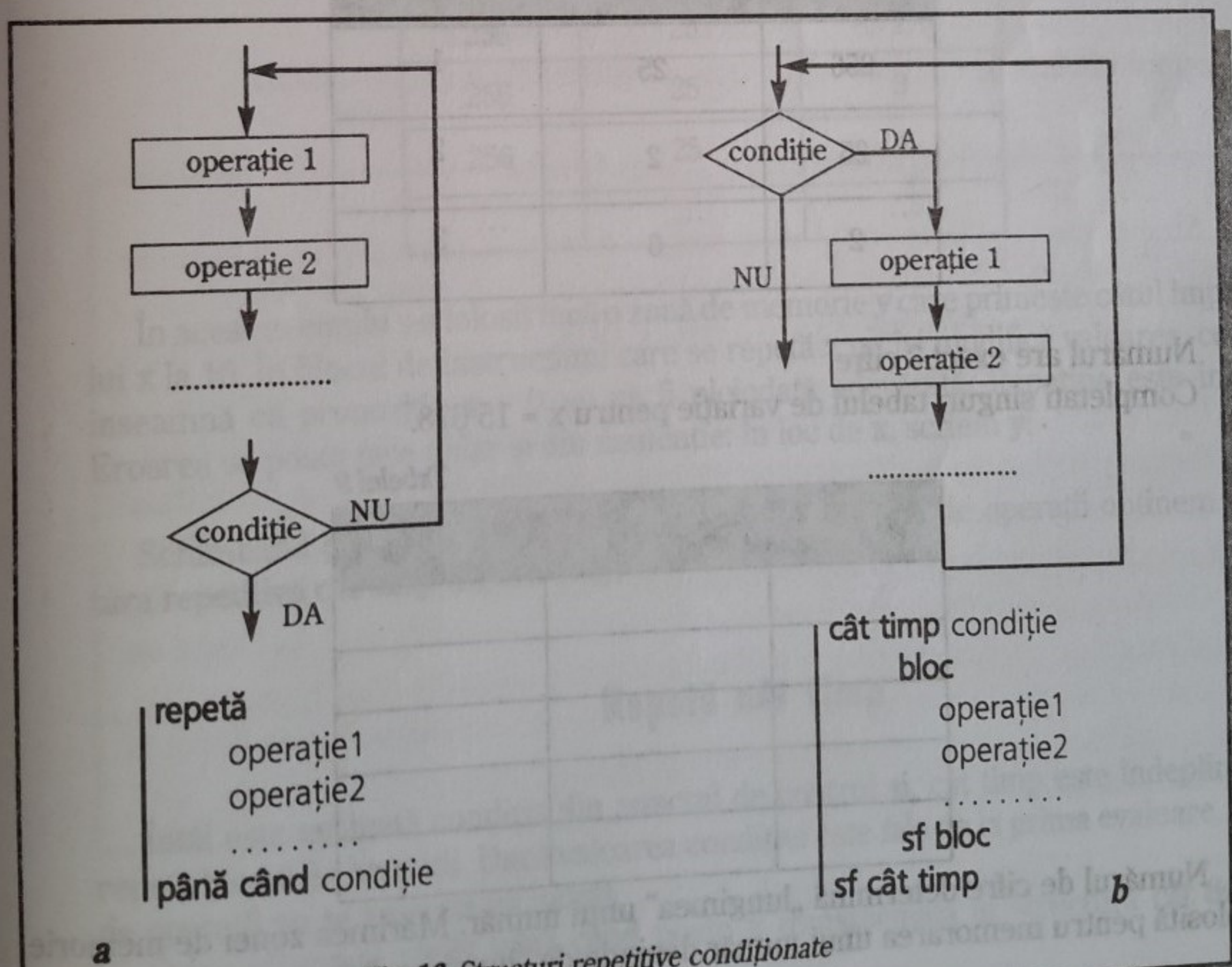


Fig. 16. Structuri repetitive condiționate

Punctul de control în care este verificată îndeplinirea condiției este plasat după secvența de operații.

Blocul de operații se execută cel puțin o dată. După prima trecere prin fiecare operație se ajunge în punctul de control și este evaluată condiția.

Dacă valoarea ei este **fals**, se repetă blocul de operații și se ajunge din nou în punctul de control. Până când? Până când valoarea condiției devine **adevărat**. Ce determină schimbarea valorii logice a condiției din **fals** în **adevărat**? Termenii prin care este formulată condiția trebuie să se modifice în blocul de operații astfel încât, la un moment dat, condiția să devină adevărată. Dacă nu are loc această transformare, condiția rămâne tot timpul pe valoarea **fals**, iar repetiția este infinită.

Spre exemplu, dorim să determinăm câte cifre are un număr întreg oarecare x introdus de la tastatură. O modalitate de rezolvare ar fi să repetăm împărțirea numărului la 10 până când câtul devine zero. Numărul de repetiții este egal chiar cu numărul de cifre pe care îl notăm cu C . Fie $x = 256$. Inițial nu știm câte cifre are numărul, deci luăm C zero.

Completăm tabelul de variație:

Tabelul 8

x	$x \text{ div } 10$	$C = 0$
256	25	1
25	2	2
2	0	3

Numărul are exact 3 cifre.

Completați singuri tabelul de variație pentru $x = 15\,678$.

Tabelul 9

x	$x \text{ div } 10$	$C = 0$

Numărul de cifre determină „lungimea” unui număr. Mărimea zonei de memorie folosită pentru memorarea unui număr depinde de „lungimea” sa.

Algoritmul pentru determinarea numărului de cifre este următorul:

```

început nr_cifre1
  citește x
  c ← 0
  repetă
    x ← x div 10
    c ← c + 1
  până când x = 0
  scrie c
  sfârșit c
sfârșit nr_cifre1

```

Să urmărim și un alt algoritm:

```

început nr_cifre2
  citește x
  c ← 0
  repetă
    y ← x div 10
    c ← c + 1
  până când x = 0
sfârșit nr_cifre2

```

Completăm tabelul de variație:

Tabelul 10

x	x div 10	C = 0
256	25	1
256	25	2
256	25	3
256	25	4
...	...	...

În acest exemplu s-a folosit încă o zonă de memorie **y** care primește câtul împărțirii lui **x** la 10. În blocul de instrucțiuni care se repetă **x** nu își modifică valoarea, ceea ce înseamnă că propoziția **x = 0** nu va fi niciodată adevărată. Repetiția este infinită. Eroarea se poate face chiar și din neatenție: în loc de **x**, scriem **y**.

Schimbând poziția punctului de control față de blocul de operații obținem structura repetitivă **cât timp** (figura 16b).

Repetă cât timp

Întâi este evaluată condiția din punctul de control și, cât timp este îndeplinită, se repetă blocul de operații. Dacă valoarea condiției este **fals** de la prima evaluare, blocul de operații nu se execută niciodată.

Orice structură **reptă până când** poate fi transformată în structură **cât timp** (și invers).

Pentru exemplul precedent rescriem algoritmul și completăm tabelul de variație.

1. Reprezentarea algoritmului

```

început nr_cifre3
  citește x
  c ← 0
  cât timp x < > 0
  | bloc
  |   x ← x div 10
  |   c ← c + 1
  | sfbloc
sf cât timp
  scrie c
sfârșit nr_cifre3
    
```

2. Tabelul de variație

Tabelul 11

x	x div 10	C = 0
256	25	1
25	2	2
2	0	3
0	—	

Analizăm rezolvarea problemei prin cele două structuri: blocul de operații este același. Condiția structurii **repetă** este formulată prin propoziția **P**: „ $x = 0$ ”. Condiția structurii **cât timp** este formulată prin propoziția **P1** = „ $x < > 0$ ”. Comparând cele două propoziții, deducem că **P1** = **NOT(P)**. Structura **cât timp** s-a obținut din structura **repetă** prin negarea condiției.

Repetiția este mama ... algoritmilor

1. Structurile repetitive se folosesc pentru repetarea execuției unui bloc de operații.
2. În structura **repetă până când**, punctul de control este plasat după blocul de operații.
3. În structura **cât timp**, punctul de control este plasat înaintea blocului de operații.
4. Pentru transformarea structurii **repetă** în structură **cât timp**, se neagă condiția.
5. Repetarea execuției unui bloc de operații se poate face prin oricare dintre structurile repetitive, la alegerea programatorului.
6. O structură repetitivă condiționată este corect construită dacă în blocul de operații există un eveniment care să schimbe valoarea logică a condiției.
7. Dacă în blocul de operații nu există un eveniment care să schimbe valoarea logică a condiției, structura repetitivă este infinită.

Teste

1. Se dă următoarea secvență de instrucțiuni:

```

citește x
repetă
    r ← x mod 10
    dacă r mod 2 = 0
        atunci scrie r
    sf dacă
    x ← x div 10
până când x = 0
    
```

1.1. Precizați ce realizează această secvență:

- a) afișează cifrele lui **x**;
- b) afișează cifrele pare ale lui **x**;
- c) verifică dacă **x** este multiplu de 10.

1.2. Transformați secvența dată folosind structura **cât timp**.

2. Se dă următoarea secvență de instrucțiuni:

```

citește y, x
cât timp y < > 0 execută
    bloc
        x ← x + y
        y ← y div 10
    sf bloc
sf cât timp
scrie x
    
```

2.1. Precizați ce valori afișează secvența dată pentru **y = 425** și **x = 32**:

- a) 32; b) 503; c) 457.

2.2. Ce valoare trebuie să aibă **y** pentru ca structura repetitivă să se execute o singură dată?

3. Se dă următoarea secvență de instrucțiuni:

```

citește x, y
dacă x < > y atunci
    repetă
        dacă x ≥ y atunci
            x ← x + y
        altfel
            y ← y - x
        sf dacă
    până când x = y
sf dacă
    
```


Apreciați cu **adevărat** sau **fals** următoarele afirmații:

- a) secvența conține o structură repetitivă corectă întrucât la intrarea în structură x este diferit de y ;
- b) secvența conține o structură repetitivă incorectă întrucât condiția de oprire ($x = y$) nu poate fi atinsă deoarece la intrarea în structură x este diferit de y ;
- c) secvența conține o structură repetitivă infinită întrucât în blocul de instrucțiuni operația $x \leftarrow x + y$ contrazice condiția $x = y$ (x crește nelimitat).

4. Care dintre următoarele secvențe de instrucțiuni numără exact 10 valori egale cu zero citite de la tastatură:

a) citește x

$c \leftarrow 0$

repetă

 dacă $x = 0$ atunci

$c \leftarrow c + 1$

 sf dacă

 citește x

până când $c = 10$

b) citește x

repetă

$c \leftarrow 0$

 dacă $x = 0$ atunci

$c \leftarrow c + 1$

 sf dacă

până când $c = 10$

c) citește x

cât timp $x = 0$

$c \leftarrow c + 1$

până când $c = 10$

5. Se dă următoarea secvență de operații:

repetă

 citește x, y

$s \leftarrow x + y$

până când $s = 100$

Precizați ce realizează această secvență:

- a) însumează un șir de valori x și y până când suma lor este 100;
- b) citește perechi de numere x, y până când suma valorilor dintr-o astfel de pereche este 100;
- c) citește valori x, y până când $S = 100$.

6. Se dă următoarea secvență de operații:

început

 citește x

 cât timp $x < > y$ execută

 bloc

 citește x

$y \leftarrow x$

$c \leftarrow c + 1$

 sf bloc

 sf cât timp

 scrie c

sfârșit

- Apreciați cu **adevărat** sau **fals** următoarele afirmații:
- secvența este greșită întrucât nu poate fi evaluată condiția $x < > y$ la prima intrare în structură (nu se cunoaște y);
 - secvența este greșită întrucât în blocul de instrucțiuni $y \leftarrow x$ contrazice condiția $x < > y$ (repetiție infinită);
 - secvența este corectă și determină numărul de valori citite x .

15. 0 înmulțire ciudată

Ionuț, elev în clasa a VI-a, a primit de la bunicul său un joc vechi cu pietricele. Bunicul i-a spus râzând că este „calculatorul” său din școala primară. Curios, Ionuț a citit cu atenție „instrucțiunile” care abia se mai deslușeau pe hârtia îngălbenită.

1. Ține minte două numere — pentru început, valori simple, să zicem 7 și 6.
2. Formează două grămezi: prima cu 7, a doua cu 6 pietricele.
3. Dacă în prima grămadă ai un număr impar de pietricele, pune în buzunar un număr de pietricele egal cu a doua grămadă.
4. Înjumătățește prima grămadă.
5. Dublează-o pe-a doua.
6. Repetă de la 3 până când prima grămadă nu mai poate fi înjumătățită.
7. Câte pietricele ai în buzunar?”

Ionuț, și mai curios, a executat „comenzile”. S-a încurcat la înjumătățire. Cum să ia din grămadă 3 pietricele și jumătate? A decis să ia „partea întreagă”: 3 pietricele. La sfârșit a numărat pietricelele din buzunar: exact 42.

A repetat jocul: 8 pietricele în prima grămadă, 5 în a doua. La sfârșit, în buzunar, 40 de pietricele.

Încă o dată: 9 pietricele în prima grămadă, 8 în a doua.

Este clar: „jocul” este într-adevăr un „calculator” ingenios care înmulțește două numere după o metodă foarte ciudată.

Neîncrezător, s-a gândit să verifice pe hârtie. Dacă e vorba doar de un truc?

Și-a făcut câte un tabel și a trecut la treabă.

Primul joc — setul de 7 și de 6 pietricele

Tabelul 12

Prima grămadă	Numărul de pietricele este impar?	A doua grămadă	Buzunarul
7	DA	6	6
3	DA	12	18
1	DA	24	42
0		48	

Al doilea joc — setul de 8 și de 5 pietricele

Tabelul 13

Prima grămadă	Numărul de pietricele este impar?	A doua grămadă	Buzunarul
8	NU	5	
4	NU	10	
2	NU	20	
1	DA	40	40
0		80	

Completați singuri tabelul pentru al treilea joc.

Al treilea joc — setul de 9 și de 8 pietricele

Tabelul 14

Prima grămadă	Numărul de pietricele este impar?	A doua grămadă	Buzunarul
9		8	

Completați și următorul tabel pentru un joc nou.

Tabelul 15

Prima grămadă	Numărul de pietricele este impar?	A doua grămadă	Buzunarul
456		275	

Ionuț este sigur că această metodă „ciudată” de înmulțire poate fi programată. Dar înainte de program, trebuie făcut algoritmul.
 „Voi folosi trei zone de memorie: x pentru prima grămadă, y pentru a doua și B pentru buzunar. Știu să aflu dacă un număr este impar; înjumătățesc, cu div ; repetiția o fac **cât timp** x poate fi înjumătățit”.
 Iată și algoritmul realizat de Ionuț:

```

început înmulțire
    citește  $x, y$ 
     $B \leftarrow 0$ 
    cât timp  $x > 0$ 
        bloc
             $r = x \bmod 2$ 
            dacă  $r = 1$  atunci  $B \leftarrow B + y$ 
            sf dacă
             $x \leftarrow x \text{ div } 2$ 
             $y \leftarrow y + y$ 
        sf bloc
    sf cât timp
    scrie  $B$ 
sfârșit înmulțire
    
```

Ionuț este foarte mulțumit. Hotărât să afle mai multe despre metodă, l-a întrebat pe bunicu'. Astfel a aflat că metoda ciudată are un nume și mai ciudat: „metoda rusească” de înmulțire. Nedumerit, Ionuț are acum și mai multe întrebări.

Dacă nu v-ați plictisit

1. Refaceți algoritmul lui Ionuț folosind o structură **repetă până când**.
2. Completați algoritmul astfel încât calculatorul să poată afișa atât valorile celor două numere cât și produsul lor.

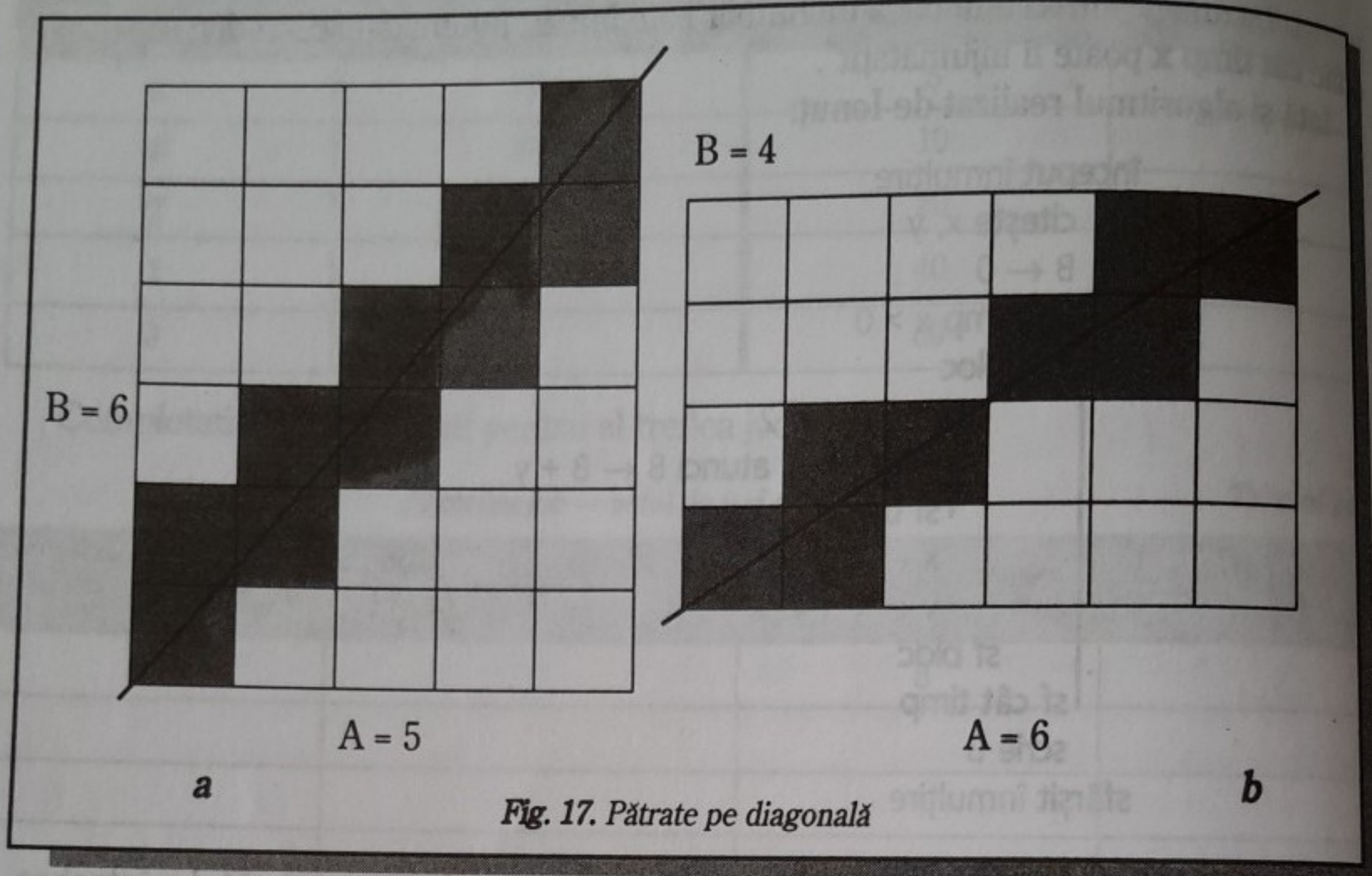
16. O problemă de geometrie... „în așteptare”

Maria este pasionată de geometrie. Nici nu-și imaginează cum s-ar putea rezolva o problemă fără desen. Pe de altă parte, este sigură că pentru calculator trebuie ceva „mai tare” decât desenul. I-am propus următoarea problemă:

„Un dreptunghi cu laturile cunoscute de A cm și B cm este împărțit în pătrate de dimensiunea 1×1 . Să se determine numărul de pătrate traversate de diagonala care unește vârfurile stânga-jos și dreapta-sus ale dreptunghiului”.

1. Analiza problemei

La început, am făcut câteva desene, ca să ne lămurim (figura 17). Apoi am completat tabelul 16.



Dimensiunile dreptunghiului		Numărul de pătrate de pe diagonală
A	B	
5	6	10
8	9	16
43	15	57
15	6	18
6	4	8
24	18	36

Valorile mari, 57 și 36 le-am completat eu. Doar nu era s-o las pe Maria să numere atâtea pătrățele!

2. Raționamentul problemei

Din desen, am observat că există două cazuri: unul în care diagonală nu trece prin colțul nici unui pătrat (figura 17a) și altul în care diagonală atinge și astfel de puncte formate la intersecția liniilor orizontale cu cele verticale din „grila” de pătrate (figura 17b).

Am studiat fiecare linie din tabel să descoperim dacă între dimensiunile laturilor și numărul de pătrate de pe diagonală există o relație care să ne ajute.

Pentru primele trei linii, Maria a descoperit relația $P = A + B - 1$.

Relația nu se verifică însă pentru celelalte cazuri. Am hotărât să cercetăm numărul dat de diferența dintre suma laturilor ($A + B$) și numărul de pătrate (P) pentru fiecare caz în parte:

Tabelul 17

A	B	$S = A + B$	P	$D = S - P$
5	6	11	10	1
8	9	17	16	1
43	15	58	57	1
15	6	21	18	3
6	4	10	8	2
24	18	42	36	6

Și tabelul pune în evidență două cazuri: cazul în care diferența dintre S și P este 1, iar numerele A și B sunt prime între ele (singurul lor divizor comun este 1), și cazul în care diferența dintre S și P este egală cu cel mai mare divizor comun al numerelor A și B (notat cmmdc_A_B). Relația căutată este $P = A + B - D$.

Așadar, cunoscând laturile A și B ale unui dreptunghi, se poate determina, fără desen, numărul de pătrate 1×1 traversate de diagonală.

1. Reprezentarea algoritmului

```

început pătrate_pe_diagonală
  citește A, B
  determină cmmdc_A_B // D
   $P \leftarrow A + B - D$ 
  scrie P
sfârșit pătrate_pe_diagonală

```

Acest algoritm conține o prelucrare insuficient descrisă: „determină cmmdc_A_B ”.

Până descoperim cum se determină cel mai mare divizor comun, problema rămâne... „în așteptare”.

17. Algoritmul lui Euclid

Deși ne-am propus să rezolvăm o problemă de geometrie fără desen, am ajuns în punctul în care soluția depinde de determinarea celui mai mare divizor comun a două numere **A** și **B**. Problema este cunoscută din antichitate și rezolvată „algoritmic” de părintele geometriei „pure” bazată pe riglă și compas — Euclid. De unde interesul matematicianului antic pentru numere prime? În antichitate, matematica, filozofia, poezia și muzica erau „practicate” cu credința că între om și natură (univers) există o armonie dată de ritmul și proporțiile dintre numere. „Totul este rânduit după număr” spunea Pythagoras (580–500 î.Hr.). Studiul numerelor a pus în evidență numerele prime (care nu au alt divizor decât 1) și o mulțime de proprietăți ale acestora.

Problema divizorilor este rezolvată de Euclid (330–275 î.Hr.) prin operații aritmetice elementare. Raționamentul este atât de simplu, de clar și de eficient, încât, toate marile dicționare contemporane ilustrează noțiunea modernă de algoritm prin această metodă moștenită de la matematicianul antic.

Vom prezenta algoritmul pentru câteva exemple concrete:

Tabelul 18

A	B	A - B	B - A	cmmdc
5	6	–	1	
	①	4	–	
4		3	–	
3		2	–	
2		1		
①				①
15	6	9	–	
9		3	–	
③			3	
	③			③
24	18	6	–	
⑥		–	12	
	12	–	6	
	⑥			⑥
6	4	2	–	
②		–	2	
	②			②

1. Care sunt pașii algoritmului?

Se compară cele două numere. Cât timp ele sunt diferite, se scade cel mai mic număr din cel mai mare. Când numerele devin egale, valoarea lor este chiar cel mai mare divizor comun.

2. Reprezentarea algoritmului

```
început euclid1
  citește A, B
  cât timp A < > B
    dacă A > B atunci A ← A-B
    altfel B ← B-A
  sf dacă
sf cât timp
scrie A
sfârșit euclid1
```

Întrucât valorile celor două numere **A** și **B** se modifică pe parcursul algoritmului, este preferabil să le mutăm în alte zone de memorie, să spunem **x** și **y**.

În acest caz, algoritmul devine:

```
început euclid2
  citește A, B
  x ← A
  y ← B
  cât timp x < > y execută
    dacă x > y atunci x ← x-y
    altfel y ← y-x
  sf dacă
sf cât timp
D ← x
scrie "cmmdc (" , A , " , " , B , ") = " , D
sfârșit euclid2
```

Comanda de scriere conține mesaje (textul cuprins între ghilimele) și valori reținute în zonele de memorie specificate.

Să ne întoarcem la problema de geometrie.

Rescriem algoritmul:

```
început pătrate-pe-diagonală
  citește A, B
  determină cmmdc_A_B // D
  P ← A + B - D
  scrie P
sfârșit pătrate-pe-diagonală
```


Introducem în rezolvare determinarea celui mai mare divizor comun. Algoritmul devine:

început pătrate-pe-diagonală

citește A, B

$x \leftarrow A$

$y \leftarrow B$

cât timp $x < > y$ execută

 dacă $x > y$ atunci $x \leftarrow x - y$

 altfel $y \leftarrow y - x$

 sf dacă

sf cât timp

$D \leftarrow x$

$P \leftarrow A + B - D$

scrie P

sfârșit pătrate-pe-diagonală

Verificați algoritmul pentru $A = 10$ și $B = 6$.

Euclidiană

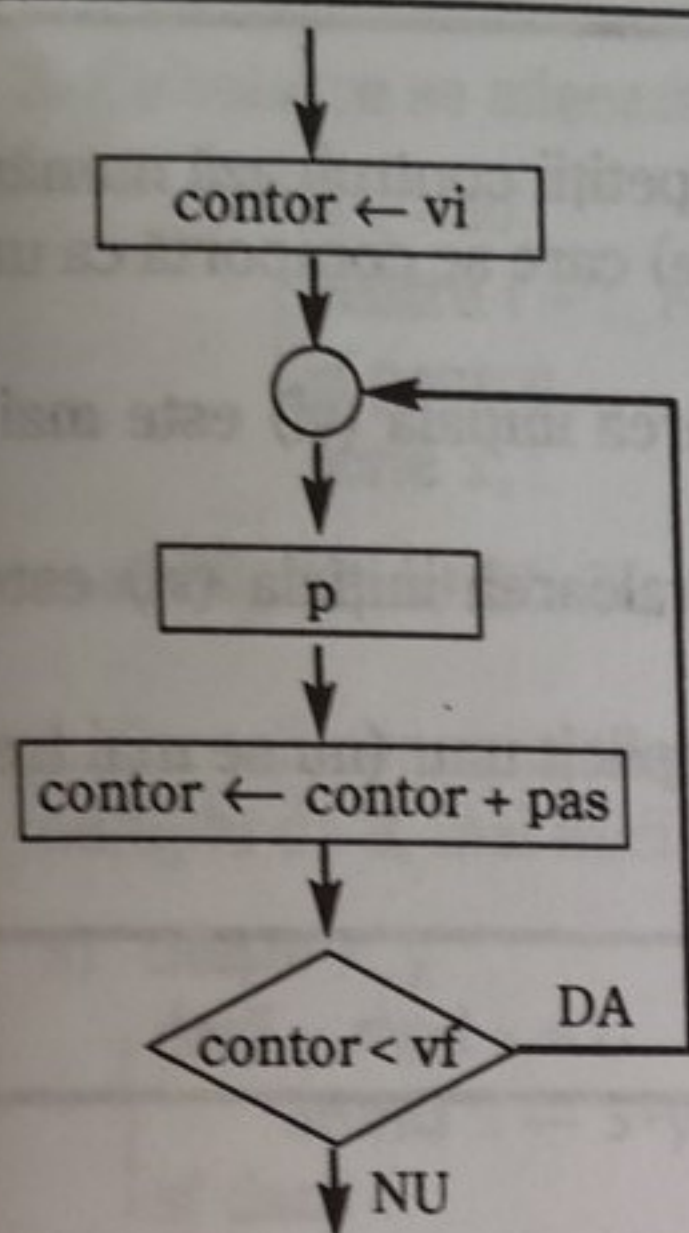
Prima universitate din lume a fost înființată în Egipt, la Alexandria, de către un urmaș al lui Ptolemeu. Biblioteca universității avea un departament de matematică condus de Euclid.

Într-o zi, un elev l-a întrebat pe Euclid la ce folosește matematica pe care o învață. După oră, Euclid s-a adresat sclavului său: „Dă-i băiatului un bănuț, de vreme ce dorește să scoată un profit din tot ceea ce învață.”. Apoi, elevul a fost exmatriculat.

18. Repetiția cu contor

Execuția unui bloc de instrucțiuni printr-un număr cunoscut de repetiții se reprezintă prin structura repetitivă cu contor **pentru** (figura 18).

Contorul, prin analogie cu cel casnic pentru măsurarea consumului electric, al apei sau al gazelor, funcționează pe principiul înregistrării prin numărare a unităților consumate (în cazul nostru, al execuției blocului de instrucțiuni). La instalare, contorul pornește de la zero. Dacă ne mutăm într-un alt apartament și păstrăm contorul vechi, pentru noi, numărătoarea pornește de la o valoare inițială (**vi**) egală cu consumul foștilor proprietari. Numărătoarea se face cu un **pas** de creștere, nu neapărat unu. Contorul crește până la o valoare finală (**vf**) cunoscută, moment în care se oprește consumul (execuția).



Pentru contor=vi, vf execută
 bloc
 :
 sf bloc
 sf pentru

Fig. 18. Structura repetitivă cu contor

La sfârșitul anului școlar, dirigintele clasei a V-a dorește să calculeze media generală a clasei. Problema se poate rezolva cu orice calculator de buzunar. Copiii s-au gândit că un program ar rezolva problema pentru toate clasele: se introduce numărul de elevi din clasă (N), apoi, pe rând, media generală (M) a fiecărui elev, după care, pe ecran se afișează media generală (MG) a clasei. Copiii au scris direct programul.

Noi vom reprezenta doar algoritmul:

început medie generală

citește N

$MG \leftarrow 0$

pentru contor = 1, N execută

bloc

citește M

$MG \leftarrow MG + M$

sf bloc

sf pentru

$MG \leftarrow MG/N$

scrie MG

sfârșit medie generală

În reprezentarea structurii prin schemă logică sunt puse în evidență toate aceste etape:

- inițializarea contorului (**contor** $\leftarrow$ vi) se realizează o singură dată;
- creșterea contorului cu valoarea pasului de numărare se execută la fiecare trecere prin blocul de instrucțiuni;
- testarea contorului față de valoarea finală.

Reprezentarea structurii în pseudocod este mult mai compactă și se apropie de structura instrucțiunilor repetitive cu contor, **FOR**, prezentă în orice limbaj de programare.

Există și situații când contorul este descrescător: spre exemplu, evidența impulsionilor telefonice gratuite. Contorul pornește de la o valoare inițială egală cu numărul de impulsuri oferite ca bonus la abonamentul telefonic și descrește cu o unitate pentru fiecare impuls consumat.

În programare, există nenumărate situații în care numărul de repetiții al unui bloc de instrucțiuni este cunoscut. De aceea, această structură este foarte mult folosită.

Refaceți algoritmul propus astfel încât N să fie o valoare validă ($N \neq 0$).

Atenție la contor!

1. Structura repetitivă cu număr cunoscut de repetiții controlează numărul de execuții cu ajutorul unei variabile (zonă de memorie) care se comportă ca un contor.

2. În cadrul repetiției cu contor crescător, valoarea inițială (**vi**) este mai mică decât cea finală (**vf**).

3. În cadrul repetiției cu contor descrescător, valoarea inițială (**vi**) este mai mare decât cea finală (**vf**).

4. Pasul de modificare al contorului, **pas**, este implicit unu (nu se mai trece în reprezentarea pseudocod).

Teste

1. Se dă următoarea secvență de operații:

```
citește N
      S ← 0
pentru i = 1, N execută
      S ← S + i
sf pentru
scrie S
```

1.1. Precizați ce realizează secvența:

- calculează numărul de execuții al structurii repetitive;
- calculează suma $S = 1 + 2 + 3 + \dots + N$;
- calculează suma primelor N numere naturale.

1.2. Propuneți un algoritm liniar pentru rezolvarea aceleiași probleme.

2. Se dă următoarea secvență de operații:

```
citește N
      P ← 1
pentru C = N, 1 execută
      P ← P * C
sf pentru
scrie P
```

2.1. Precizați ce realizează această secvență:

- calculează produsul primelor N numere naturale;
- calculează numărul de execuții al structurii repetitive;
- calculează produsul $P = N * (N - 1) * \dots * 4 * 3 * 2 * 1$.

2.2. Transformați secvența folosind structuri repetitive condiționate.

3. Ce valoare se afișează după execuția următoarei secvențe:

$N \leftarrow 30$

pentru $i = 1, N$ execută $x \leftarrow i$

sf pentru

scrie x, i

a) 30, 30; b) 31, 31; c) 30, 31; d) 29, 30.

4. Care dintre următoarele secvențe de operații determină suma numerelor y , multiplu de x , mai mici decât 100?

a) citește x, y

dacă $y \bmod x = 0$

atunci $S \leftarrow S + y$

sf dacă

dacă $S < 100$

atunci citește x, y

sf dacă

scrie S

b) citește x

$P \leftarrow x$

dacă $x < 100$ atunci

repetă

$P \leftarrow P * x$

$S \leftarrow S + P$

până când $P > 100$

scrie S

sf dacă

c) citește x

$P \leftarrow x$

cât timp $P < 100$ execută

bloc

$S \leftarrow S + P$

$P \leftarrow P * x$

sf bloc

sf cât timp

scrie S

d) citește x

$P \leftarrow x$

$S \leftarrow 0$

cât timp $P < 100$ execută

bloc

$P \leftarrow P * x$

$S \leftarrow S + P$

sf bloc

sf cât timp

scrie S

5. Se dă următoarea secvență de operații:

citește x, y

$S \leftarrow 0$

pentru $i = x, y$ execută

$S \leftarrow S + i$

sf pentru

scrie S

5.1. Pentru $x = 10$ se afișează valoarea 75. Ce valoare are y ?

5.2. Pentru ce valori ale lui x și y se afișează valoarea zero?

5.3. Propuneți un algoritm liniar pentru rezolvarea aceleiași probleme?

6. Ce realizează următoarea secvență de operații:

```
repetă citește x până când  $x < 10$   
pentru  $i = 1, 10$  execută  
scrie  $i * x$   
sf pentru
```

- a) scrie de 10 ori $x * i$;
- b) afișează valori de forma $1 * x, 2 * x, 3 * x, \dots, 10 * x$;
- c) afișează tabla înmulțirii cu x .

19. Trasee turistice

Elevii clubului ecologic „Floare de colț” pregătesc o nouă acțiune de asigurare a traseelor turistice montane. De această dată vor instala corzi în zonele cu risc mare de accidentare. Pentru pregătirea acțiunii, au nevoie de harta zonei turistice. Instructorul le propune un alt mod de lucru: vor primi descrierea traseului sub forma unui șir de numere care reprezintă cotele punctelor aflate în observație. Întrucât corzile de siguranță sunt foarte scumpe, hotărâsc să le monteze doar între punctele a căror diferență de nivel este mai mare de 5 m. În prima etapă a acțiunii, instructorul vrea să știe numărul minim de corzi pe care le vor instala și cât de mare poate fi cea mai lungă dintre corzi.

Exemplu: Pentru un traseu cu 10 puncte în observație, având cotele montane 3, 7, 15, 20, 28, 32, 46, 50, 53 și 59 sunt necesare 4 corzi, cea mai lungă fiind de 14 m.

1. Analiza problemei

- Date de intrare
 - numărul de puncte (N)
 - N valori reprezentând cotele (H)
- Date de ieșire
 - numărul de corzi (C)
 - coarda cea mai lungă (L)

- Condiții și relații importante

Se calculează diferența D dintre două puncte consecutive H_2 și H_1 : $D = H_2 - H_1$. Interesează cazul în care $D > 5$.

2. Raționamentul problemei

P1. Se introduce numărul de puncte, N .

P2. Se citesc primele două valori, H_1 și H_2 .

- P3. Se calculează D și se verifică condiția.
 P4. Dacă este îndeplinită condiția, crește numărul de corzi și lungimea corzii instalate se memorează în L .
 P5. Se păstrează valoarea din $H2$ în $H1$.
 P6. Se citește o nouă valoare în $H2$.
 P7. Se calculează D și se verifică condiția.
 P8. Dacă este îndeplinită condiția, crește numărul de corzi și se compară ultima coardă cu cea memorată. În memorie se păstrează valoarea cea mai mare.
 P9. Se repetă operațiile 5, 6, 7, 8 pentru toate valorile.
 P10. Se afișează numărul de corzi, C , și coarda cea mai lungă, L .

3. Reprezentarea algoritmului

început traseu montan1

citește N

citește $H1, H2$

$C \leftarrow 0$

$L \leftarrow 0$

$D \leftarrow H2 - H1$

dacă $D > 5$ atunci

bloc

$C \leftarrow C + 1$

$L \leftarrow D$

sf bloc

sf dacă

pentru $i \leftarrow 1, N-2$ execută

bloc

$H1 \leftarrow H2$

citește $H2$

$D \leftarrow H2 - H1$

dacă $D > 5$ atunci

bloc

$C \leftarrow C + 1$

dacă $D > L$ atunci $L \leftarrow D$

sf dacă

sf bloc

sf dacă

sf bloc

sf pentru

scrie C, L

sfârșit traseu montan1

4. Verificarea algoritmului

Pentru verificarea algoritmului urmăriți și completați tabelul de variație:

Tabelul 18

N	H1	H2	D	D > 5	C	L	
10	3	7	4	NU	0	0	
	7	15	8	DA	1	8	1
	15	20	5	NU			2
	20	28	8	DA	2		3
	28	32	4	NU			4

Modificați algoritmul astfel încât să nu mai fie necesare operațiile **P3** și **P4**.

De reținut!

În rezolvarea acestei probleme s-a folosit raționamentul pentru **determinarea valorii maxime** dintr-un șir de numere.

20. Organizarea datelor

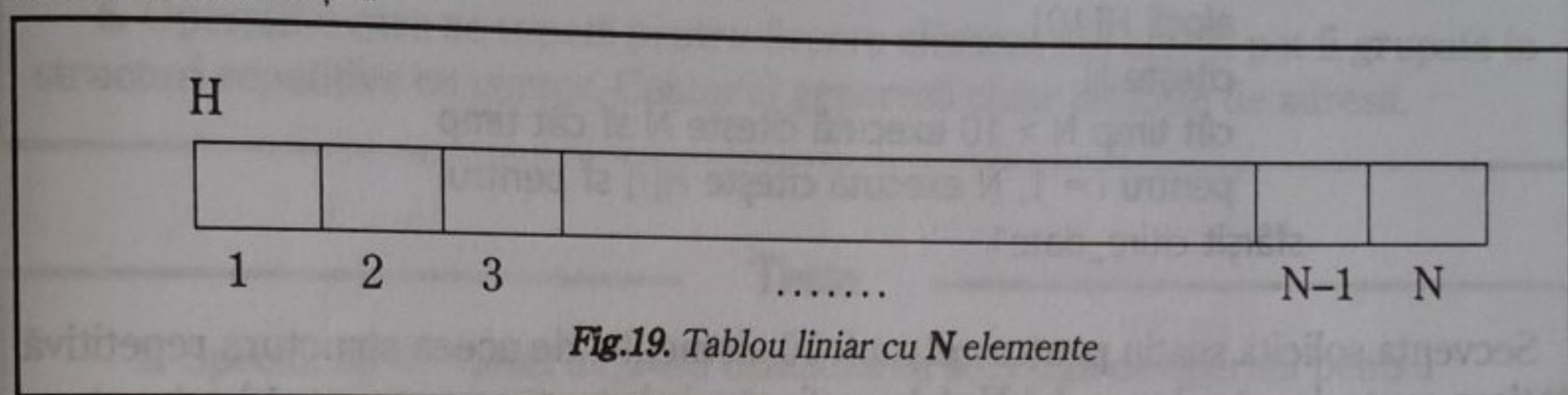
În problema traseelor montane sunt prelucrate mai multe valori cu aceeași semnificație: cotele montane ale punctelor din traseu. Rezolvarea problemei a răspuns strict la cerința determinării numărului de corzi și a corzii de lungime maximă. De aceea, a fost suficientă păstrarea în memorie, la un moment dat, doar a ultimelor două valori consecutive **H1** și **H2**. Dar dacă instructorul ar dori și alte prelucrări, de genul: câte corzi de aceeași lungime **H** se vor folosi sau câte corzi egale între ele se vor instala? Rezolvarea unor astfel de cerințe necesită păstrarea tuturor valorilor în memoria calculatorului. Ar fi necesare **N** zone de memorie: **H1**, **H2**, **H3**, ..., **Hn**. Doar pentru introducerea datelor ar fi necesare **N** operații distincte de citire, câte una pentru fiecare zonă de memorie: citește **H1**, citește **H2**, ... Valoarea lui **N** variază de la un traseu la altul, de la o execuție la alta. Ar trebui să adăugăm sau să ștergem comenzi

înaintea fiecărei execuții. Ar fi un chin care ne-ar lecui, cu siguranță, de plăcerea de a programa.

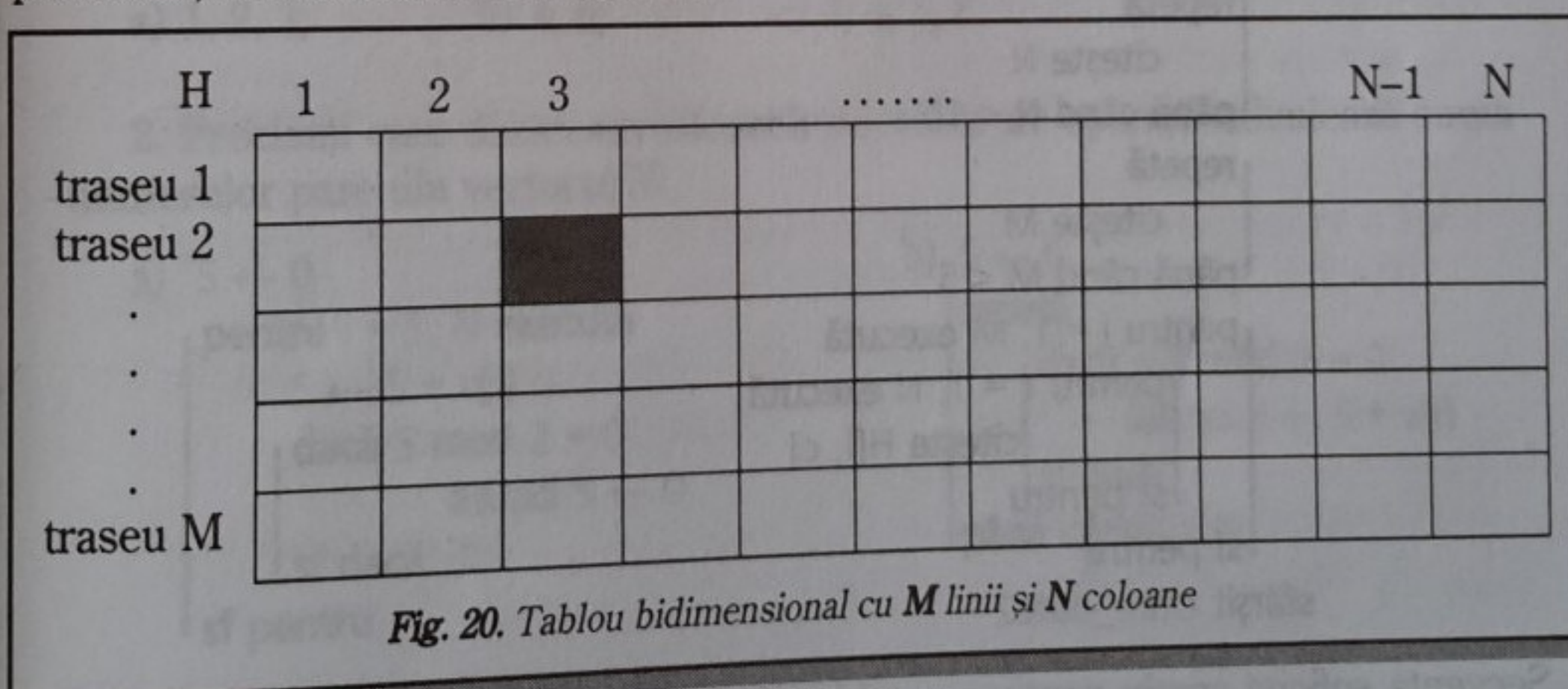
Situația pe care o analizăm se rezolvă organizând datele cu aceeași semnificație într-un ansamblu numit **tablou**. Ansamblul primește un nume și fiecare element are o adresă compusă din numele tabloului și poziția elementului în tablou ca o adresă poștală cu stradă și număr sau bloc și apartament.

Pentru cele N cote montane, tabloul s-ar putea numi H , iar adresele elementelor ar fi $H[1]$, $H[2]$, ... $H[N]$.

Intuitiv, spunem că un astfel de tablou are o formă liniară. Elementele sunt dispuse pe o singură dimensiune. Altfel spus, elementele sunt organizate după un singur criteriu: semnificația.



Dacă dorim să păstrăm datele pentru mai multe trasee, să spunem M ? Fiecare traseu are N puncte. Tabloul necesită $M \times N$ elemente. Adresând un element, să spunem cel de pe poziția 100 în tablou, pierdem semnificația reală a valorii sale: cărui traseu îi aparține și cărui punct din traseu. În acest caz, ar fi util să organizăm tabloul pe trasee, fiecare traseu cu N puncte.



Un astfel de tablou are două dimensiuni, datele fiind organizate pe linii (trasee) și coloane (puncte). Adresa unui element din tablou se compune din numele tabloului împreună cu linia și coloana pe care e situat elementul. De exemplu, $H[2, 3]$ semnifică elementul aflat în tabloul H pe linia 2, coloana 3.

O zonă de memorie organizată ca tablou poate fi solicitată din orice limbaj de programare. Încă din faza de proiectare a algoritmului, programatorul trebuie să

stabilească capacitatea tabloului (numărul maxim de elemente în cazul tabloului liniar; numărul maxim de linii și coloane în cazul tabloului bidimensional).

Calculatorul pune la dispoziția programatorului (**alocă**) tabloul de capacitate maximă. În timpul execuției, vor fi folosite efectiv doar locațiile necesare potrivit valorilor **N** și **M**. Identificarea elementelor prin adresă permite realizarea de secvențe scurte de operații care se repetă, pentru fiecare element al tabloului, doar prin schimbarea adresei.

Spre exemplu, citirea și memorarea cotelor montane în tabloul liniar se face prin secvența:

```
început citire_date1
    alocă H[10]
    citește N
    cât timp N > 10 execută citește N sf cât timp
    pentru i = 1, N execută citește H[i] sf pentru
sfârșit citire_date1
```

Secvența solicită spațiu pentru cel mult 10 elemente, de aceea structura repetitivă **cât timp** controlează valoarea lui **N**. Adresa fiecărui element se generează în structura repetitivă cu contor și este dată chiar de valoarea contorului.

Citirea și memorarea datelor într-un tablou bidimensional se poate face prin secvența:

```
început citire_date2
    alocă H[5, 10]
    repetă
        citește N
        până când N < 10
    repetă
        citește M
        până când M < 5
        pentru l = 1, M execută
            pentru c = 1, N execută
                citește H[l, c]
            sf pentru
        sf pentru
    sf pentru
sfârșit citire_date2
```

Secvența solicită spațiu pentru un tablou bidimensional cu cel mult 5 linii și 10 coloane.

Structurile **repetă până când** controlează valorile lui **N**, respectiv **M**.

Elementele din tablou sunt citite în ordinea liniilor, de aceea prima structură cu contor generează adresa de tip linie. Adresa de tip coloană este generată de a doua structură cu contor. Operația de citire se execută de **M * N** ori, exact numărul de elemente din tablou.

Cum organizăm datele în tablouri

1. Dacă într-o problemă este necesară memorarea mai multor valori cu aceeași semnificație, aceste valori vor fi grupate într-un ansamblu de tip **tablou**.
2. În funcție de cerințele problemei și de semnificația datelor, acestea sunt organizate în tablouri cu o singură dimensiune, numite **vectori**, sau în tablouri cu două dimensiuni, numite **matrice**.
3. Elementele unui tablou se identifică printr-o adresă formată din numele tabloului și câte un indice pentru fiecare dimensiune.
4. La tablourile cu două dimensiuni, primul indice din adresă reprezintă linia, iar cel de-al doilea coloana tabloului.
5. Operațiile care se repetă pentru fiecare element din tablou pot fi grupate în structuri repetitive cu contor. Contorul generează chiar indicele de adresă.

Teste

1. Specificați ce valori afișează următoarea secvență de operații pentru $V = 0, 3, 2, 4, 1, 6, 8$:

```
pentru i = 1, 7 execută
    dacă v[i] = i atunci scrie i
    sf dacă
sf pentru
```

- a) 1, 2, 3; b) 4, 6; c) 2, 4, 6, 8.

2. Precizați care dintre următoarele secvențe de operații calculează suma numerelor pare din vectorul V :

a) $S \leftarrow 0$

```
pentru i = 1, N execută
     $S \leftarrow S + v[i]$ 
    dacă  $S \bmod 2 = 0$ 
        atunci  $S \leftarrow 0$ 
    sf dacă
sf pentru
```

b) $S \leftarrow 0$

```
repetă
    dacă  $v[i] \bmod 2 = 0$ 
        atunci  $S \leftarrow S + v[i]$ 
    sf dacă
până când  $i > N$ 
```

c) $S \leftarrow 0$

```
pentru i = 1, N execută
    dacă  $v[i] \bmod 2 = 0$ 
        atunci  $S \leftarrow S + v[i]$ 
    sf dacă
sf pentru
```


3. Dați exemple de valori pentru care următoarea secvență de operații afișează valoarea 3:

```
z ← 0
pentru i = 1, N execută
    dacă v[i] = 0
        atunci z ← z + 1
    sf dacă
sf pentru
scrie z
```

4. Precizați ce realizează următoarea secvență de operații:

```
alocă x[10], y[10]
pentru i = 1, 10 execută
    citește x[i], y[i]
sf pentru
n ← 0
pentru i = 1, 10 execută
    dacă x[i] = y[i] atunci n ← n + 1
    sf dacă
sf pentru
scrie n
```

a) afișează numărul de elemente ale vectorilor **x** și **y**;

b) verifică dacă cei doi vectori, **x** și **y**, sunt egali;

c) determină și afișează numărul de elemente egale în cei doi vectori.

5. Ce valori afișează următoarea secvență de operații pentru vectorul $V = 2, 6, 4, 2, 5, 7, 1, 8, 2, 6$

```
pentru i = 1, 5 execută
    bloc
        v[i] ← v[i] + i
        scrie v[i]
    sf bloc
sf pentru
```

a) 2, 6, 4, 2, 5; b) 3, 7, 5, 3, 6; c) 1, 2, 3, 4, 5.

6. Se consideră tabloul **M** cu următoarele elemente:

1	2	3	4
5	6	7	8
9	10	11	12

Precizați ce valori afișează secvența de mai jos:

```

pentru c = 1, 3 execută
    pentru l = 1, 2 execută
        scrie M[l, c]
    sf pentru
sf pentru
    
```

a) 1, 5, 9, 2, 6, 10, 3, 7, 11; b) 1, 2, 3, 5, 6, 7; c) 1, 5, 2, 6, 3, 7.

7. Ce realizează următoarea secvență de operații:

```

alocă M[10, 10]
pentru i = 1, 10 execută
    M[i, i] ← i
sf pentru
    
```

- a) atribuie valori de la 1 la 10 elementelor din vectorul **M**;
- b) atribuie fiecărui element de pe diagonala matricei **M** o valoare egală cu linia pe care acesta se află;
- c) atribuie valori de la 1 la 10 primelor 10 elemente din matricea **M**.

8. Care dintre următoarele secvențe de operații memorează în colțurile matricei **M** [5, 5] valori citite de la tastatură:

```

a) pentru i = 1, N execută
    pentru j = 1, N execută
        citește M[i, j]
    sf pentru
sf pentru
    
```

```

b) pentru i = 1, 5 execută
    pentru j = 1, 5 execută
        dacă (i = 1) și (j = 5)
            atunci citește M[i, i]
            altfel citește M[j, j]
        sf dacă
    sf pentru
sf pentru
    
```

```

c) l ← 1
   c ← 5
   pentru x = 1, 4 execută
       citește M[l, c]
       c ← l
       l ← c
   sf pentru
    
```

```

d) l ← 1
   c ← 5
   citește M[l, l], M[l, c], M[c, l], M[c, c]
    
```


21. Din nou trasee turistice

Reluăm problema traseelor turistice în scopul rezolvării prin memorarea cotelor montane într-un vector **H**. Raționamentul de rezolvare rămâne același. Elementele consecutive vor fi de forma **H[i]** și **H[i - 1]**.

1. Reprezentarea algoritmului

```
început traseu_montan2
    alocă H[10]
    citește N
    cât timp N > 10 execută citește N sf cât timp
    pentru i = 1, N execută citește H[i] sf pentru
    C ← 0
    L ← 0
    pentru i = 2, N execută
        bloc
            D ← H[i] - H[i - 1]
            dacă D > 5 atunci
                bloc
                    C ← C + 1
                    dacă D > L atunci
                        L ← D
                    sf dacă
                sf bloc
            sf dacă
        sf bloc
    sf pentru
    scrie C, L
sfârșit traseu_montan2
```

Membrii clubului „Floare de colț” participă la o tabără de vară, timp de o săptămână, într-o zonă montană greu accesibilă. Scopul lor este să înregistreze în fiecare zi temperatura aerului și să determine temperatura minimă, temperatura maximă și zilele în care s-au atins aceste temperaturi.

1. Analiza problemei

- *Date de intrare*

7 valori reprezentând temperatura înregistrată în fiecare zi a săptămânii

- *Date de ieșire*

- temperatura minimă (**tmin**);
- temperatura maximă (**tmax**);
- ziua/zilele în care s-au atins **tmin** și **tmax**.

2. Organizarea datelor

Datele de intrare au aceeași semnificație și vor fi înregistrate într-un vector (**T**) cu 7 elemente.

De exemplu:

T	18	15	12	14	18	15	12
----------	----	----	----	----	----	----	----

tmin = 12 înregistrată miercuri și duminică

tmax = 18 înregistrată luni și vineri

3. Raționamentul problemei

Se înregistrează temperaturile zilnice în vectorul **T**. Se determină, printr-o singură parcurgere, valoarea minimă și cea maximă.

Pentru afișarea zilei/zilelor se mai fac două parcurgeri ale vectorului. Ziua va fi afișată ca indice (1, 2 etc.). Pentru afișarea zilei prin nume (luni, marți etc.) se recomandă introducerea structurii selective după **zi**.

4. Reprezentarea algoritmului

```
început temperaturi1
alocă T[7]
pentru zi = 1, 7 execută citește T[zi] sf pentru
tmin ← T[1]
tmax ← T[1]
pentru zi = 2, 7 execută
    bloc
        dacă T[zi] < tmin atunci tmin ← T[zi]
        sf dacă
        dacă T[zi] > tmax atunci tmax ← T[zi]
        sf dacă
    sf bloc
sf pentru
scrie tmin
pentru zi = 1, 7 execută
    dacă T[zi] = tmin atunci scrie zi
    sf dacă
sf pentru
scrie tmax
pentru zi = 1, 7 execută
    dacă T[zi] = tmax atunci scrie zi
    sf dacă
sf pentru
sfârșit temperaturi1
```

Măsurarea temperaturii aerului în zonele greu accesibile a fost foarte apreciată de ecologiști. Numărul voluntarilor dornici să participe la astfel de acțiuni a crescut. Instructorul clubului a organizat zece echipe care să măsoare temperatura zilnică în diverse zone de interes.

Prelucrarea valorilor înregistrate se va face astfel:

- se afișează temperatura minimă și temperatura maximă înregistrată în cele zece zone pe parcursul săptămânii;
- se afișează temperatura medie, pentru oricare dintre zone, la solicitarea celui interesat;
- se afișează zona în care s-a atins cea mai mică, respectiv cea mai mare temperatură, pentru orice zi a săptămânii solicitată de cel interesat.

1. Analiza problemei

Datele problemei au aceeași semnificație – temperatura zilnică –, dar se referă la zone diferite; de aceea, pentru organizarea lor se introduce și criteriul **zonă**. Rezultă

Zona	T						
	1	2	3	4	5	6	7
1						21	
2						18	
3	18	15	12	14	18	15	12
4						16	
5						17	
6						19	
7						20	
8						20	
9						17	
10						18	
	1	2	3	4	5	6	7
	Ziua						

un tablou (**T**) cu două dimensiuni: **zonă** (linie) și **ziua** (coloană).

2. Raționamentul problemei

Pentru determinarea temperaturii minime și a temperaturii maxime înregistrate în cele zece zone, pe parcursul săptămânii, se prelucrează toate elementele din tablou.

Pentru determinarea temperaturii medii, **tmed**, se solicită zona și se prelucrează doar elementele de pe linia corespunzătoare.

Pentru determinarea zonei în care s-a atins temperatura minimă sau maximă, se solicită ziua și se prelucrează doar coloana corespunzătoare.

3. Reprezentarea algoritmului

```
început temperaturi2
alocă T[10, 7]
pentru zona = 1, 10 execută
    pentru zi = 1, 7 execută
        citește T[zona, zi]
    sf pentru
sf pentru
tmin ← T[1, 1]
tmax ← T[1, 1]
pentru zona = 1, 10 execută
    pentru zi = 1, 7 execută
        dacă T[zona, zi] < tmin
            atunci tmin ← T[zona, zi]
        sf dacă
        dacă T[zona, zi] > tmax
            atunci tmax ← T[zona, zi]
        sf dacă
    sf pentru
sf pentru
scrie tmin
scrie tmax
scrie "ce zonă vă interesează?"
citește zona
tmed ← 0
pentru zi = 1, 7 execută
    tmed ← tmed + T[zona, zi]
sf pentru
tmed ← tmed/7
```

```
scrie zona, tmed
scrie "ce zi vă interesează?"
citește zi
tmin ← T[1, zi]
zmin ← 1
tmax ← T[1, zi]
zmax ← 1
pentru zona = 2, 10 execută
    bloc
        dacă T[zona, zi] < tmin
            atunci
                bloc
                    tmin ← T[zona, zi]
                    zmin ← zona
                sf bloc
        sf dacă
        dacă T[zona, zi] > tmax
            atunci
                bloc
                    tmax ← T[zona, zi]
                    zmax ← zona
                sf bloc
        sf dacă
    sf bloc
sf pentru
scrie zmin, tmin, zi
scrie zmax, tmax, zi
sfârșit temperaturi2
```

22. Ciurul lui Eratostene

Geograful și astronomul grec Eratostene din Cirena (284–192 î.Hr.) a fost preocupat și de numerele cu proprietăți deosebite, cum ar fi șirul numerelor prime: 2, 3, 5, 7, 11, 13, ... Studiind acest șir, Eratostene a descoperit o metodă „practică” pentru obținerea numerelor prime mai mici decât un număr dat. Întrucât un număr prim nu este multiplul nici unuia dintre numerele mai mici decât el, Eratostene s-a gândit să elimine pe rând toți multipli lui 2, 3, 5, 7 etc. mai mici decât numărul dat. Procedeu de selectare a numerelor prime folosit de Eratostene seamănă cu o cernere: „ciurul” folosește „site” pe „măsura” numerelor prime. Mulțimea valorilor inițiale este cernută până când în ciur rămân doar numere prime.

Spre exemplu, dorim să obținem toate numerele prime mai mici decât 25. În ciur sunt așezate inițial toate numerele: 1, 2, 3, ..., 24, 25. Prima cernere se face cu sita lui 2 și se elimină toți multipli acestuia: 4, 6, 8, ..., 22, 24. Urmează cernerea cu sita lui 3 și se elimină 9, 15, 21 (multiplii pari ai lui 3 au fost eliminați).

Procedeul continuă pentru 5, 7 etc. La sfârșit, ciurul conține numerele prime mai mici decât 25: 2, 3, 5, 7, 11, 13, 19, 23.

Procedeul poate fi algoritmitizat.

Fie N numărul de numere naturale din care facem selecția numerelor prime. Așezăm numerele într-un tablou de tip vector. Vectorul va fi „ciurul” nostru.

Eliminarea — „cernerea” — se va face prin parcurgerea vectorilor din 2 în 2, din 3 în 3, din 5 în 5 etc. și înscrierea valorii zero pe poziția multiplului astfel eliminat.

La sfârșit, singurele valori diferite de zero din vector vor fi numerele prime, mai mici decât N , pe care le afișăm.

1. Reprezentarea algoritmului

început ciur_eratostene

 alocă $C[100]$

 repetă

 citește N

 până când $N < 100$

 pentru $k=2, N$ execută $C[k] \leftarrow k$

 sf pentru

$p \leftarrow 2$

$k \leftarrow 4$

 repetă

 repetă

$C[k] \leftarrow 0$

$k \leftarrow k + p$

 până când $k > N$

$p \leftarrow p + 1$

 cât timp $C[p] = 0$ execută $p \leftarrow p + 1$

 sf cât timp

$k \leftarrow p + p$

 până când $k > N$

 pentru $k=2, N$ execută

 dacă $C[k] < > 0$ scrie $C[k]$

 sf dacă

 sf pentru

sfârșit ciur_eratostene

2. Verificarea algoritmului

Verificați algoritmul completând tabelul de variație pentru $N = 25$, $N = 50$.

Tabelul 19

k	p	C[k]
4	2	0

Modificați algoritmul transformând structurile **repetă până când** în structuri repetitive cu contor.

Modificați algoritmul astfel încât să folosiți doar poziția numerelor în ciur (nu valoarea lor).

23. Viruși în doi vectori

Biologii studiază o colonie de bacterii infectate cu diverși viruși. O bacterie purtătoare transmite virusul către una sau mai multe bacterii. Este căutat virusul cel mai puternic. Bacteria purtătoare îl poate transmite tuturor bacteriilor din grup, dar ea nu poate fi atacată de nici un alt virus. În fișa de laborator, biologii au notat numărul de bacterii din grup (fiecare bacterie este infectată cu un virus) și modul de transmitere a virușilor între două bacterii: de la bacteria x la bacteria y .

Exemplu: Colonia studiată are 5 bacterii. Transmiterea virușilor se face între bacteriile: (2, 1), (2, 3), (5, 3), (4, 1), (5, 4), (1, 3), (5, 2), (5, 1), (3, 2).

Virusul cel mai rezistent este cel purtat de bacteria 5: el se răspândește la toate celelalte, iar bacteria 5 nu primește nici un virus (vezi figura 19).

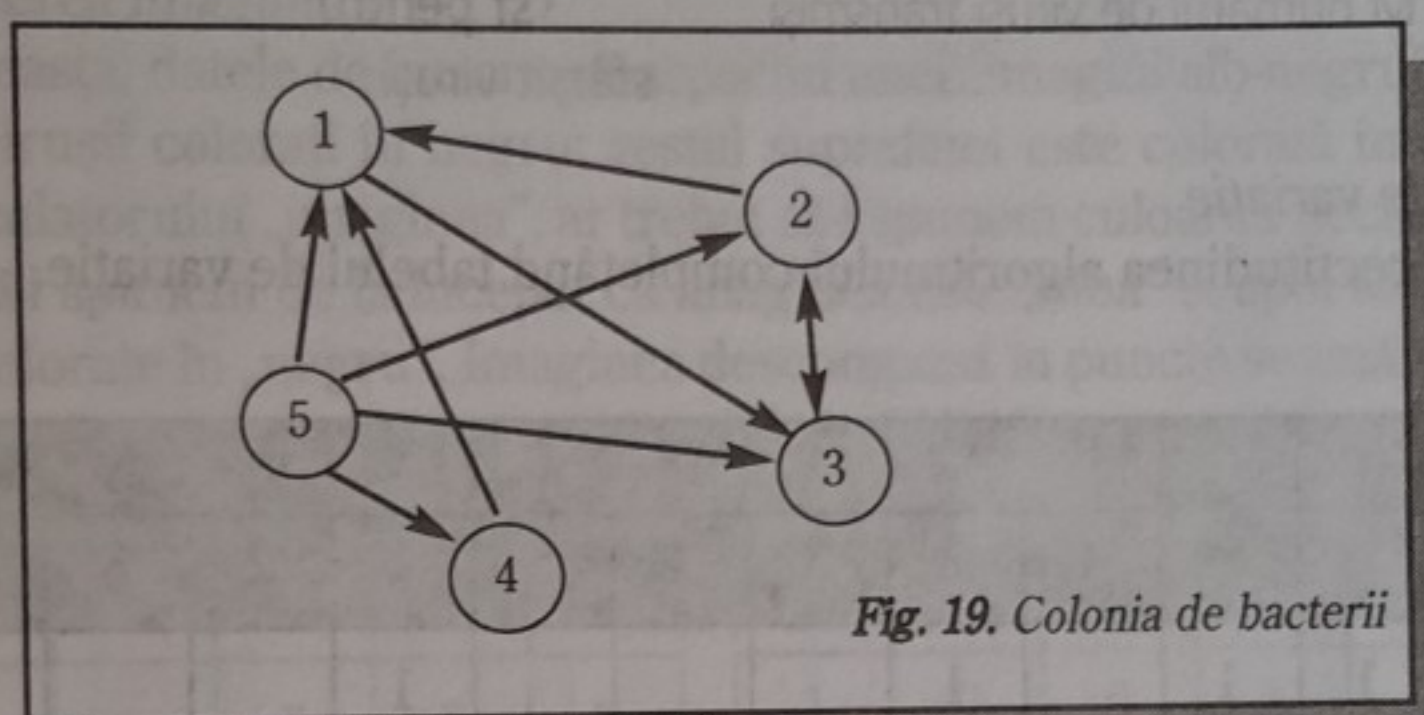


Fig. 19. Colonia de bacterii

1. Cum rezolvăm problema cu calculatorul?

Ar trebui să numărăm câți viruși transmite (t) fiecare bacterie și câți primește (p). Apoi căutăm bacteria din care au plecat $N - 1$ viruși și care n-a primit nici unul.

Rezolvarea pe care o propunem folosește doi vectori: X pentru numărarea virușilor transmiși și Y pentru numărarea virușilor primiți.

x		• •			•
	1	2	3	4	5
y	•		• •		

Se înregistrează toți virușii transmiși: (2, 1), (2, 3), (5, 3). Completați singuri cei doi vectori pentru celelalte perechi de bacterii de forma (t, p) din exemplul dat.

La sfârșit, se parcurge vectorul X și, pentru elementul cu valoarea $N - 1$ (dacă acesta există), se verifică dacă elementul aflat pe aceeași poziție în Y are valoarea zero.

2. Reprezentarea algoritmului

început viruși

alocă $X[10]$, $Y[10]$

repetă

citește N

până când $N < 10$

pentru $c = 1, N$ execută

bloc

$X[c] \leftarrow 0$

$Y[c] \leftarrow 0$

sf bloc

sf pentru

citește M numărul de viruși transmiși

pentru $c = 1, M$ execută

bloc

citește t, p

$X[t] \leftarrow X[t] + 1$

$Y[p] \leftarrow Y[p] + 1$

sf bloc

sf pentru

pentru $c = 1, N$ execută

dacă $(X[c] = N - 1)$ și $(Y[c] = 0)$
atunci scrie c

sf dacă

sf pentru

sfârșit viruși

3. Tabelul de variație

Verificați corectitudinea algoritmului completând tabelul de variație.

Tabelul 20

N	t	p	c	X					Y				
				1	2	3	4	5	1	2	3	4	5
5	2	1	1		1				1				
	2	3			2						1		

Organizarea datelor în tablouri permite memorarea valorilor de același fel și stabilirea de relații între acestea.

Scopul acestei probleme a fost acela de a ilustra utilizarea vectorilor pentru prelucrarea relațiilor dintre elemente.

24. Viruși într-o matrice

După ce au descoperit cei mai puternici viruși, biologii și-au propus să-i izoleze și să le studieze comportamentul în alte colonii de bacterii. După o vreme, au constatat că unii viruși au rămas izolați, alții nu. Privită la microscop, colonia de bacterii pare formată dintr-o mulțime de puncte pe care biologii le-au reprezentat ca în figura 20.

În această colonie există un singur virus izolat (cel încercuit în figura 20). Biologii doresc să prelucereze aceste „imagini” cu calculatorul. Pentru început vor să localizeze și să numere virușii izolați.

Cum rezolvăm problema?

1. Introducerea imaginii

De data aceasta, datele de intrare au aspectul unei „imagini alb-negru”: punctele de interes sunt virușii colorați în negru; restul suprafeței este colorată în alb. Pentru a transmite calculatorului „imaginea”, ar trebui să-i spunem culoarea fiecărui punct. Sau mai simplu, să-i spunem de la început că imaginea este „albă” și apoi să-i dăm adresele punctelor colorate în „negru”. Imaginea descompusă în puncte seamănă atât de bine cu o matrice încât ea poate fi memorată codificând albul cu zero și negrul cu unu.

	x				x
x			x	x	x
			x	x	
	(x)		x		

0	1	0	0	0	1
1	0	0	1	1	1
0	0	0	1	1	0
0	1	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Fig. 20. Localizarea virușilor izolați

2. Reprezentarea algoritmului

Secvența de operații pentru citirea imaginii:

început citire

alocă M[10,10]

citește NL, NC // număr linii -NL
// număr coloane -NC

pentru i = 1, NL execută

pentru j = 1, NC execută

M[i, j] ← 0

sf pentru

sf pentru

citește P // numărul de puncte negre

pentru k = 1, P execută

bloc

citește l, c

M[l, c] ← 1

sf bloc

sf pentru

sfârșit citire

3. Prelucrarea imaginii. Bordajul

Urmează prelucrarea „imaginii”: localizarea și numărarea virușilor (punctelor negre izolate).

Pentru rezolvarea acestei cerințe se parcurge matricea și, pentru fiecare valoare de unu, se verifică vecinii acesteia. Un punct „negru” este izolat dacă toți vecinii săi sunt „albi”.

Câți vecini are un punct? Majoritatea punctelor au câte opt vecini. Punctele de la „bariera” coloniei au mai puțini vecini.

Pentru a simplifica procedeul de numărare a vecinilor vom înconjura colonia cu o zonă fără viruși. Imaginea se lărgeste. La fel și matricea: în urma acestui „bordaj”, ea se va mări cu două linii și două coloane.

0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	1	1	1	0
0	0	0	0	1	1	0	0
0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Bordajul se pregătește înainte de citirea imaginii prin alocarea unui spațiu de memorie suficient ținând seama că la numărul de linii și coloane necesar memorării imaginii se mai adaugă câte două pentru bordaj.

În matricea bordată, fiecare punct al imaginii are câte opt vecini. În figura 21 sunt prezentate adresele vecinilor punctului aflat la adresa $[l, c]$.

$l-1, c-1$	$l-1, c$	$l-1, c+1$
$l, c-1$	l, c	$l, c+1$
$l+1, c-1$	$l+1, c$	$l+1, c+1$

Fig. 21. Vecinii unui punct din matrice

Dacă punctul aflat la adresa $[l, c]$ are valoarea unu și fiecare dintre cei opt vecini ai săi are valoarea zero, punctul este izolat. Condiția o vom nota cu **cond**.

4. Reprezentarea algoritmului

- Secvența pentru localizarea și numărarea punctelor izolate:

```
început puncte_izolate
nrp ← 0
pentru l = 2, NL + 1 execută
    pentru c = 2, NC + 1 execută
        dacă M[l, c] = 1 atunci
            dacă cond
                atunci
                    bloc    scrie l - 1, c - 1
                           nrp ← nrp + 1
                sf bloc
            sf dacă
        sf dacă
    sf pentru
sf pentru
scrie nrp
sfârșit puncte_izolate
```

Temă

Folosind cele două secvențe prezentate, realizați algoritmul complet.

De reținut!

1. Cu ajutorul matricelor pot fi rezolvate probleme care necesită descompunerea unei imagini în puncte sau probleme de orientare și deplasare în plan.
2. Pentru ușurința prelucrării punctelor se folosește **tehnica bordajului**.
3. Multe jocuri precum „x și zero”, „vapurașul”, „perspico”, mutarea pieselor pe tabla de șah, „război” pot fi simulate cu ajutorul matricelor.

25. Rucsacul și algoritmul lacom

Nu vă grăbiți să închideți cartea. Nu plecăm în excursie. Ne mutăm. Ne mutăm în alt capitol al programării, mai tehnic și plin de metode.

Pentru început, trebuie să golim o încăpere plină cu tot felul de obiecte de diferite mărimi, greutateți și chiar utilități. Cred că n-au mai fost folosite de mult. De aceea ni se și cere să golim repede camera, din cât mai puține mutări. Avem la dispoziție doar un rucsac care este destul de încăpător, dar nu „ține” mai mult de 30 kg. Obiectele au fost cântărite cândva și pe fiecare este înscrisă greutatea (în kilograme): 7, 8, 10, 25, 5, 7, 8, 10, 9, 5, 6, 7, 25, 5. Ce se întâmplă dacă nu golim camera? Un algoritm, scuzați, căpcăun flămând și lacom așteaptă la ușă. Avem voie să ne facem un plan. Mai ales că aceasta e abia prima încăpere. Mai sunt și altele. Și toate trebuie golite cu rucsacuri de diverse capacități.

O modalitate ar fi să luăm obiectele la rând, să le încărcăm repede în rucsac și gata! Fiecare rucsac, o mutare. În cazul nostru:

Rucsac	Mutări
$7 + 8 + 10 = 25$	1
$25 + 5 = 30$	2
$7 + 8 + 10 = 25$	3
$9 + 5 + 6 + 7 = 27$	4
25	5
5	6

E clar că nu e cea mai bună soluție. Ultima mutare e pentru un rucsac aproape gol. Obiectul de 5 kg ar fi încăput și-n primul rucsac, și-n al treilea, dacă era „la rând”.

Ce-ar fi să „rânduim” altfel obiectele? Într-o ordine care să ne convină nouă. Să nu ne trezim, la sfârșit, că mai avem de făcut o mutare pentru doar 5 kg. Mai ales că vom fi obosiți. Cum nu contează decât greutatea, să așezăm obiectele, la rând, în această ordine: 5, 5, 5, 6, 7, 7, 7, 8, 8, 9, 10, 10, 25, 25. Să vedem câte mutări sunt necesare acum:

Rucsac	Mutări
$5 + 5 + 5 + 6 + 7 = 28$	1
$7 + 7 + 8 + 8 = 30$	2
$9 + 10 + 10 = 29$	3
25	4
25	5

Am redus o mutare și-am obținut numărul minim cu puțință, în acest caz, de 5 mutări. Se poate verifica: suma greutateilor este 130 kg, capacitatea rucsacului este de 30 kg; sunt necesare 5 rucsacuri.

Căpcăunul flămând a adormit, așa că putem folosi și calculatorul să ne punem la punct planul.

Pentru calculator vom înlocui încăperea cu un vector. Pentru început, punem obiectele în ordinea din cameră. Apoi le așezăm după greutate și, în cele din urmă, le „adunăm” în rucsac — o zonă de memorie a cărei valoare nu va fi mai mare de 30.

Deși nu e terminat, algoritmul arată astfel:

```
început rucsac
  alocă G[100]
  repetă citește N până când N < 100
  pentru i=1,N execută citește G[i]
  sf pentru
  ordonează G
  R ← 0 //golim rucsacul
  m ← 0
  i ← 1
  repetă
    dacă G[i] ≤ 30 - R
      atunci
        bloc
          R ← R + G[i]
          i ← i + 1
        sf bloc
      altfel
        bloc
          m ← m + 1
          R ← 0
        sf bloc
    sf dacă
  până când i > N
  m ← m + 1
  scrie m //numărul de rucsacuri
sfârșit rucsac
```

Algoritmul e pregătit să golească o cameră cu cel mult 100 de obiecte folosind un rucsac cu capacitate de 30 kg. Pentru generalizare, se poate citi și capacitatea rucsacului, Q.

Mai lacom decât căpcăunul de la ușă, algoritmul nostru abia așteaptă să „înghită” obiectele din rucsac; numele său este **Greedy**.

Despre Greedy

1. Metodă de programare bazată pe aranjarea (ordonarea) convenabilă a datelor astfel încât să se obțină o soluție sigură și optimă.

2. Rucsacul este și el un obiect „clasic” în programare. În foarte multe împrejurări se cere încărcarea unei capacități (rucsac) cu cât mai multe obiecte sau cu obiecte cât mai utile.

3. Problema rucsacului se rezolvă prin metoda **Greedy** ordonând convenabil „obiectele”: crescător (pentru cât mai multe) sau descrescător (pentru cât mai utile).

26. Cum ordonăm datele

Pentru a umple eficient rucsacul, obiectele au fost reasezate în ordinea de la cel mai ușor la cel mai greu.

Rezolvarea problemei cu calculatorul nu este încă posibilă întrucât nu am prezentat algoritmul prin care se obține această reasezare a elementelor.

Inițial, vectorul greutăților, **G**, avea următoarele elemente:

G	7	8	10	25	5	7	8	10	9	5	6	7	25	5
----------	---	---	----	----	---	---	---	----	---	---	---	---	----	---

Prin ordonare, elementele și-au schimbat poziția astfel încât la stânga oricărui element se află toate valorile mai mici decât el, iar la dreapta sa, toate valorile mai mari:

G	5	5	5	6	7	7	7	8	8	9	10	10	25	25
----------	---	---	---	---	---	---	---	---	---	---	----	----	----	----

Trecerea vectorului de la configurația inițială (vector neordonat) la configurația finală (vector ordonat) se face prin compararea elementelor învecinate și schimbarea poziției lor astfel încât valorile mici să se deplaseze spre stânga, iar cele mari spre dreapta.

Pentru exemplificare, vom urmări ordonarea unui vector cu mai puține elemente:

7	8	10	25	5
---	---	----	----	---

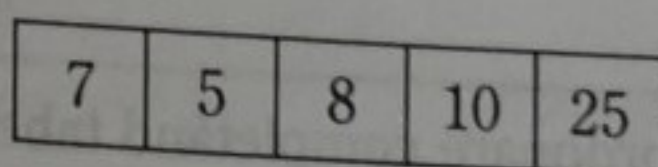
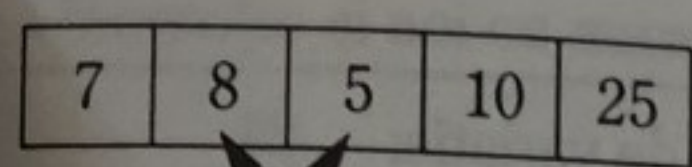
7	8	10	5	25
---	---	----	---	----

← prima parcurgere

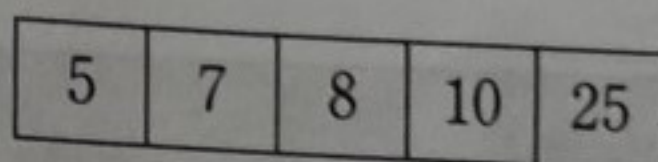
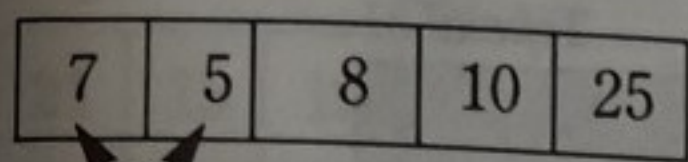
7	8	10	5	25
---	---	----	---	----

7	8	5	10	25
---	---	---	----	----

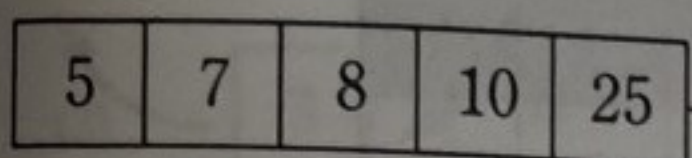
← a 2-a parcurgere



← a 3-a parcurgere



← a 4-a parcurgere



← a 5-a parcurgere

Vectorul a fost parcurs de mai multe ori până când s-a ajuns la configurația finală. La fiecare parcurgere s-au comparat rând pe rând elementele învecinate și s-a făcut schimbarea de poziții (interschimbul) doar între valorile care nu respectau relația de ordine: 25 cu 5 în prima parcurgere, 10 cu 5 la a doua, 8 cu 5 la a treia și 7 cu 5 la a patra. Configurația finală se recunoaște prin faptul că nu s-a mai făcut nici un interschimb.

1. Raționamentul metodei

Se repetă parcurgerea vectorului cu interschimbul elementelor care nu respectă relația de ordine până când se ajunge la o parcurgere fără interschimb.

Pentru a reține dacă a avut loc sau nu cel puțin un interschimb într-o parcurgere, algoritmul va folosi o zonă de memorie **S** cu două valori posibile: 1, dacă a avut loc un interschimb și 0, dacă nu a avut loc nici un interschimb.

2. Reprezentarea algoritmului

început ordonează-G

repetă

$S \leftarrow 0$

pentru $i=1, N-1$ execută

dacă $G[i] > G[i+1]$ atunci

bloc

$A \leftarrow G[i+1]$

$G[i+1] \leftarrow G[i]$

$G[i] \leftarrow A$

$S \leftarrow 1$

sf bloc

sf dacă

sf pentru

până când $S=0$

sfârșit ordonează-G

3. Tabelul de variație

Verificați algoritmul de ordonare completând tabelul de variație:

Tabelul 21

N	S	i	A	G				
				1	2	3	4	5
5	0	1		7	8	10	25	5
		2						
		3						
		4	5				5	25

Secvența de ordonare poate fi introdusă în algoritmul oricărei probleme.

Modelarea bulelor

1. Ordonarea datelor prin interschimb și „mișcarea” lor spre stânga (valorile mici) sau dreapta (valorile mari), asemenea unor bule, poartă numele de **Bubble Sort** sau **Metoda bulelor**.
2. Metoda necesită un timp de lucru care depinde de numărul de treceri prin vector și de numărul de interschimburi la fiecare trecere.
3. Există numeroase metode de sortare a căror eficiență se măsoară după timpul de execuție și memoria folosită.
4. Orice programator trebuie să cunoască metodele de sortare și să aleagă folosirea unei metode în funcție de criteriul de eficiență urmărit.

27. Dacă ești mic, stai în față!

Nu vă repeziți să ordonați obiectele din cameră! Poate găsim și altă soluție. Vă mai amintiți cum vă așeza doamna învățătoare după înălțime? Îl scotea din rând pe cel mai mic: „dacă ești mic, stai în față!”. Primul copil din rând fugea pe locul celui mai mic și acesta se așeza pe prima poziție. Apoi, venea în față cel mai mic dintre copiii rămași. Și tot așa, până la ultimii doi.

Să încercăm și noi cu numere:

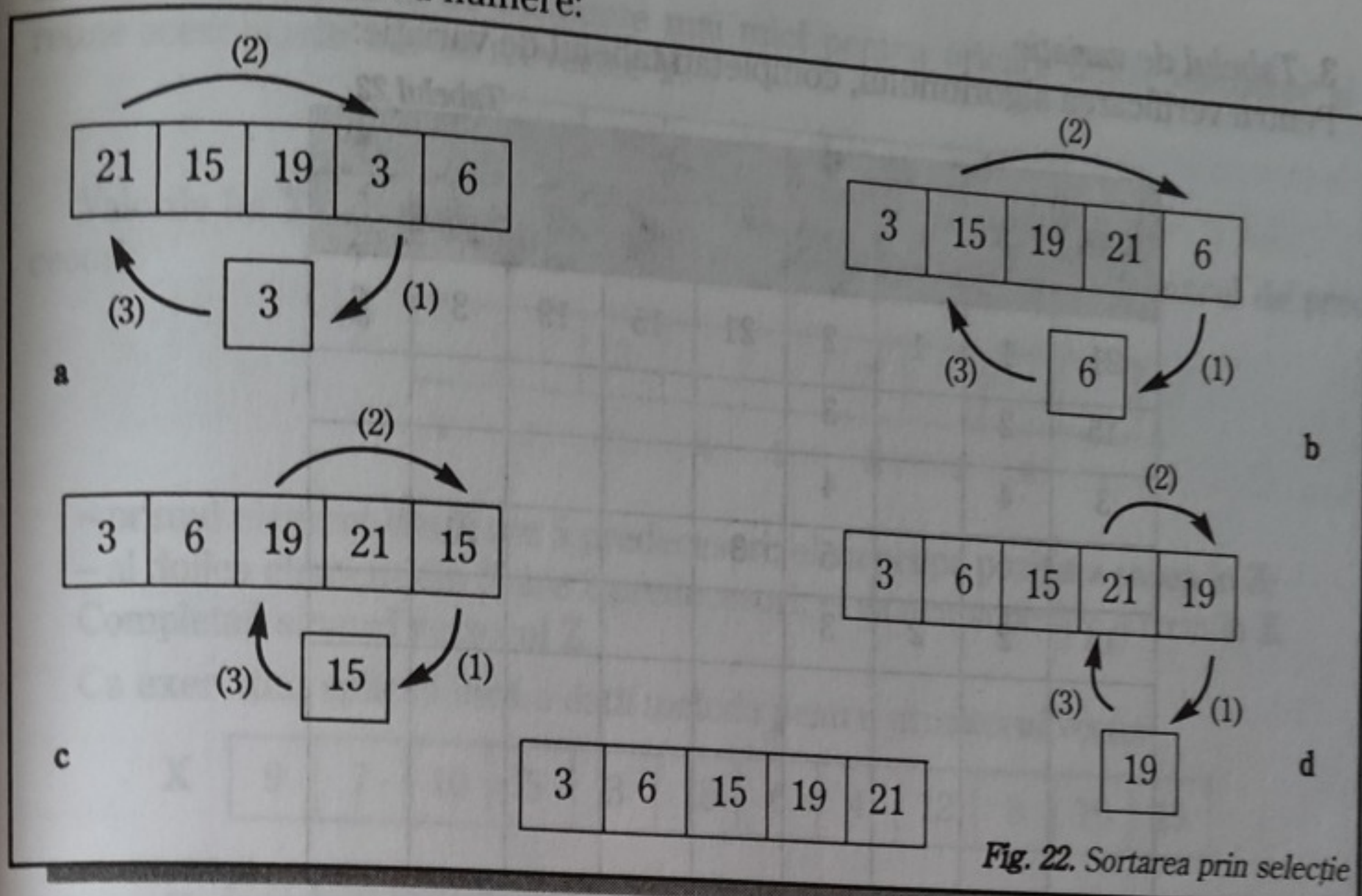


Fig. 22. Sortarea prin selecție

1. Raționamentul metodei

Pentru calculator, șirul de copii va fi un vector C . Determinăm elementul minim (MIN) și realizăm interschimbul cu cel aflat pe prima poziție (p); pentru interschimb, reținem adresa elementului minim (m). Șirul în care se caută elementul minim devine tot mai scurt, până ajunge la doar două elemente. Pentru reprezentarea algoritmului considerăm că vectorul este citit și are N elemente.

2. Reprezentarea algoritmului

început ordonare2

pentru $p=1, N-1$ execută

bloc

$MIN \leftarrow C[p]$

$m \leftarrow p$

pentru $i=p+1, N$ execută

dacă $C[i] < MIN$ atunci

bloc

$MIN \leftarrow C[i]$

$m \leftarrow i$

sf bloc

sf dacă

sf pentru

$C[m] \leftarrow C[p]$

$C[p] \leftarrow MIN$

sf bloc

sf pentru

sfârșit ordonare2

3. Tabelul de variație

Pentru verificarea algoritmului, completați tabelul de variație:

Tabelul 22

MIN	m	p	i	C				
				1	2	3	4	5
21	1	1	2	21	15	19	3	6
15	2		3					
3	4		4					
			5	3			21	
15	2	2	3					

Sortarea prin selecție

1. Metoda alegerii celui mai mic element dintr-un vector și așezarea lui pe prima poziție, repetată pentru șiruri din ce în ce mai scurte, se numește **sortare prin selecție** sau **Selectsort**.

2. Metoda necesită un timp de lucru care depinde de numărul de elemente (N) din vector.

3. Algoritmul metodei se reprezintă prin structuri repetitive cu număr cunoscut de pași.

28. Numărarea predecesorilor

Fiecare element al unui șir ordonat crescător are un număr fix de predecesori (elemente aflate înaintea sa). Dacă știm numărul de predecesori, știm implicit poziția elementului în vectorul ordonat.

Folosind această observație, vom prezenta o altă metodă de ordonare: prin numărare. Ce numărăm? Elementele mai mici față de fiecare element al vectorului.

1. Raționamentul metodei

Pentru vectorul **X** cu următoarele valori

X	7	3	9	2	10	1	8	5	4
----------	---	---	---	---	----	---	---	---	---

se determină numărul de elemente mai mici pentru oricare dintre elemente și se reține acest număr într-un alt vector **Y**.

Y	5	2	7	1	8	0	6	4	3
----------	---	---	---	---	---	---	---	---	---

Valorile lui **X** vor fi copiate într-un vector **Z** pe poziția dată de numărul de predecesori:

Z			3			7			
	1	2	3	4	5	6	7	8	9

- primul element din **X** are 5 predecesori; el va ocupa poziția a șasea în **Z**;
 - al doilea element din **X** are 2 predecesori; el va ocupa poziția a treia în **Z**.
- Completați singuri vectorul **Z**.

Ca exercițiu, aplicați încă o dată metoda pentru următorul vector:

X	9	7	10	5	3	18	6	4	12	8	16	15
----------	---	---	----	---	---	----	---	---	----	---	----	----

Y												
----------	--	--	--	--	--	--	--	--	--	--	--	--

Z												
----------	--	--	--	--	--	--	--	--	--	--	--	--

2. Reprezentarea algoritmului

```

început ordonare3
    alocă X[100], Y[100], Z[100]
    repetă
        citește N
    până când N < 100
    pentru i=1,N execută citește X[i] sf pentru
    pentru i=1,N-1 execută
        pentru j=i+1,N execută
            dacă X[i] < X[j] atunci Y[j] ← Y[j]+1
            altfel Y[i] ← Y[i]+1
        sf dacă
    sf pentru
    sf pentru
    pentru i=1,N execută
        bloc
            k = Y[i]+1
            Z[k] ← X[i]
        sf bloc
    sf pentru
    pentru i=1,N execută scrie Z[i] sf pentru
sfârșit ordonare3
    
```


Ordonarea prin numărare parcurge de un număr fix de ori vectorul inițial și folosește încă doi vectori: unul pentru numărarea predecesorilor, celălalt pentru obținerea vectorului sortat.

3. Tabelul de variație

Pentru verificarea algoritmului, completați tabelul de variație:

Tabelul 23

N	i	j	X				Y				Z			
			1	2	3	4	1	2	3	4	1	2	3	4
4			7	3	5	8								
	1	2					1							
		3					2							
		4							1					
	2	3												

29. O alegere dificilă

Suntem în situația de a alege unul dintre mai mulți algoritmi care rezolvă aceeași problemă. Cum alegem cel mai bun algoritm? După ce criteriu? Aliniem algoritmi la „start” și, cel care ajunge primul la linia de sosire (datele de ieșire), este declarat cel mai rapid.

Așadar, măsurăm timpul de execuție. Un bun programator nu-și poate permite să rezolve o problemă prin mai multe metode și, la sfârșit, să măsoare timpul de execuție ca să decidă cea mai bună soluție. E ca și cum, spunându-i unui taximetrist că vrem să ajungem foarte repede la adresa **x**, acesta ne-ar plimba până acolo pe trei trasee diferite ca să poată stabili drumul cel mai scurt. Cine suportă costul? Asemeni taximetristului care apreciază „la comandă” drumul cel mai scurt, în funcție de traseu și de trafic, programatorul își pune problema determinării timpului de execuție chiar în faza proiectării algoritmului. Este vorba de un timp teoretic întrucât, practic, timpul de execuție depinde de viteza de lucru a procesorului.

Timpul teoretic în care un algoritm rezolvă problema depinde de numărul de operații executate.

Pentru a compara între ei doi algoritmi, se determină **operațiile de bază**, al căror număr de execuții variază de la un algoritm la altul; în general, acestea sunt **operații de atribuire** și de **comparare**. Contează și numărul de valori (**n**) cărora li se aplică aceste operații.

Exemplu:

- În secvența pentru citirea unui vector **V** cu **n** elemente, operația de citire se execută de **n** ori:

```

| pentru i = 1, n execută
|     citește V [i]
| sf pentru

```

- În secvența pentru citirea unei matrice **M** cu **n** linii și **n** coloane, operația de citire se execută de **n x n = n²** ori:

```

| pentru l = 1, n execută
|     | pentru c = 1, n execută
|     |     citește M [l, c]
|     | sf pentru
|     sf pentru

```

- În secvența pentru determinarea tripletelor **a, b, c** mai mici decât un **n** dat, **a < b < c**, care respectă relația **c² = a² + b²**, operațiile de atribuire și verificare a relației se execută de **n * n * n = n³** ori:

```

| pentru a = 1, n execută
|     | pentru b = a+1, n execută
|     |     | pentru c = b+1, n execută
|     |     |     bloc
|     |     |     a ← a*a
|     |     |     b ← b*b
|     |     |     c ← c*c
|     |     |     dacă c = a+b atunci scrie a, b, c
|     |     |     sf dacă
|     |     sf bloc
|     sf pentru
| sf pentru

```

Din exemplele prezentate, rezultă că numărul de execuții al operațiilor de bază se poate exprima în forma **O(n)**, **O(n²)**, **O(n³)**, formă numită și **ordinul de mărime** sau **complexitatea** algoritmului (vezi tabelul 24).

Se consideră ca fiind cel mai bun, algoritmul cu ordinul de mărime cel mai mic.

În faza de proiectare, programatorul determină complexitatea algoritmului și caută să reducă ordinul de mărime pentru a rezolva problema într-un timp cât mai scurt.

Complexitate	Tipul de algoritm
$O(n)$	algoritm liniar
$O(n^2)$	algoritm pătratic
$O(n^3)$	algoritm cubic
$O(2^n)$	algoritm exponențial

În exemplul precedent, complexitatea secvenței pentru determinarea tripletelor a , b , c poate fi redusă în mod evident de la $O(n^3)$ la $O(n^2)$:

```

a ← 1
cât timp a ≤ 98 execută
    bloc
        b ← a+1
        cât timp b ≤ 99 execută
            bloc
                c ← a*a + b*b
                c ← √c
                dacă (c < 100) și (c*c=a*a + b*b)
                    atunci
                        scrie a, b, c
                sf dacă
                b ← b+1
            sf bloc
        sf cât timp
        a ← a+1
    sf bloc
sf cât timp

```

Pentru același algoritm, complexitatea poate varia în funcție de relațiile dintre valorile datelor de intrare.

Spre exemplu, algoritmul de sortare **Bubble Sort** are complexitatea $O(n)$ dacă secvența datelor de intrare este gata ordonată (vectorul este parcurs o singură dată; nu e necesar nici un interschimb) și complexitatea $O(n^2)$ în rest.

Cazul care conduce la complexitatea cea mai mică este numit **situație favorabilă**. Se caută, însă, și cazul care conduce la complexitatea cea mai mare (sau la numărul cel mai mare de operații executate pentru aceeași complexitate) —acesta corespunde **situației defavorabile**. Restul cazurilor sunt considerate **situații intermediare** sau **medii**.

Ca exercițiu practic, să analizăm metodele de sortare prezentate punând în evidență numărul de parcurgeri, comparații și atribuiri în funcție de gradul de sortare al datelor de intrare. Precizăm că o astfel de analiză se face pentru n foarte mare.

Analiza comparativă a metodelor de sortare

Tabelul 25

Metoda de Situția sortare analizată	Bubble Sort	Select Sort	Numărarea predecesorilor
Situția favorabilă 1, 4, 5, 7, 8, 9	– o singură parcurgere – doar comparații – nici un interschimb $O(n)$	– parcurgeri repetate – doar comparații – număr minim de atribuiri $O(n^2)$	
Situția defavorabilă 9, 8, 7, 5, 4, 1	– parcurgeri repetate – număr maxim de interschimburi $O(n^2)$	– parcurgeri repetate – comparații și interschimburi – număr maxim de atribuiri $O(n^2)$	număr fix de parcurgeri, comparații și atribuiri, indiferent de gradul de sortare $O(n^2)$
Situția medie 5, 7, 1, 8, 4, 9	– parcurgeri repetate – numărul de interschimburi depinde de gradul de sortare $O(n^2)$	– parcurgeri repetate – comparații și interschimburi – numărul de atribuiri depinde de gradul de sortare $O(n^2)$	

Complexitatea algoritmilor

1. Complexitatea reprezintă un criteriu de apreciere a eficienței unui algoritm.

2. Timpul de execuție al unui algoritm este cu atât mai mic cu cât complexitatea acestuia este mai redusă.

3. Complexitatea algoritmului este aproximată prin numărul operațiilor de bază executate (atribuiri, comparații) și se notează cu $O(f(n))$, unde $f(n)$ este relația care dă ordinul de mărime.

30. Deviza cuceritorului

Vă mai amintiți de căpcăunul cel lacom? Are un fiu foarte mofturos. Ar mânca întruna, orice. Nimic nu-i este, însă, pe plac. Astăzi i s-a făcut poftă de obiectele din cameră. Și, pentru că nu știe cum se numesc, urlă în gura mare: „vreau 10 kg” sau „vreau 6 kg” sau „vreau 80 kg”. Căpcăunul-fiu este totuși copil. Nu are voie să înghită mai mult de un obiect o dată. Vă dați seama ce ne așteaptă? Să controlăm prin cameră și să găsim rapid chiar obiectul de 10 kg sau pe cel de 6 kg. S-ar putea ca un astfel de obiect să nu existe în cameră și, dacă nu-i spunem la timp, ne trezim în fălcile căpcăunului-tată.

Trebuie să găsim o soluție rapidă. Să folosim calculatorul!

Obiectele sunt reținute în vectorul **G**.

Vectorul **G** este ordonat crescător:

G	5	5	5	6	7	7	7	8	8	9	10	10	25	25
----------	---	---	---	---	---	---	---	---	---	---	----	----	----	----

N-avem decât să-i spunem calculatorului ce greutate căutăm, iar el să răspundă cu poziția pe care se află obiectul sau cu un mesaj, dacă nu există un astfel de obiect.

Cum află calculatorul poziția obiectului?

O cale sigură ar fi să verifice pe rând greutatea fiecărui obiect, până ajunge la greutatea cerută. În camerele cu foarte multe obiecte, căutarea va dura însă prea mult. Să reducem numărul de verificări!

Vom proceda în felul următor:

- verificăm întâi dacă are rost să căutăm sau nu obiectul: greutatea să nu fie mai mică decât a primului obiect din vector sau mai mare decât a ultimului obiect;
- alegem un obiect reper – cel din mijloc – și comparăm greutatea căutată cu greutatea acestuia; s-ar putea să nimerim chiar obiectul căutat;
- dacă greutatea căutată este mai mare decât a obiectului din mijloc, e clar că-l vom căuta printre obiectele aflate la dreapta acestuia; altfel, îl vom căuta la stânga;
- căutarea la dreapta sau la stânga o vom face la fel, alegând ca reper obiectul din mijloc.

Să verificăm dacă „merge”:

G	5	5	5	6	7	7	7	8	8	9	10	10	25	25
	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Căpcăunul vrea 10 kg. Cel mai ușor obiect din cameră are 5 kg, cel mai greu are 25 kg. Începem căutarea:

- localizăm obiectul reper pe poziția 7;
- comparăm greutatea căutată cu greutatea obiectului din mijloc: $10 > 7$; rezultă că obiectul căutat s-ar putea afla la dreapta; continuăm căutarea:

8	8	9	10	10	25	25
8	9	10	11	12	13	14

- localizăm obiectul reper pe poziția 11;
- comparăm greutatea căutată cu cea a obiectului reper: $10 = 10$; rezultă că obiectul căutat a fost găsit pe poziția 11 din doar două căutări.

1. Raționamentul metodei

Pentru scrierea algoritmului de căutare, folosim următoarele notații:

- **p** și **u** pentru adresa primului, respectiv ultimului element din secvența de valori în care se face, la un moment dat, căutarea. În exemplul nostru, la prima căutare, $p = 1$ și $u = 14$; la a doua căutare, $p = 8$ și $u = 14$;

- **m** pentru adresa elementului din mijlocul secvenței în care se face căutarea:

$$m = (p + u) \text{ div } 2$$

În exemplul nostru, la prima căutare, $m = (1 + 14) \text{ div } 2 = 7$, iar la a doua căutare, $m = (8 + 14) \text{ div } 2 = 11$.

Inițial, **p** și **u** au valorile 1, respectiv 14. După ce se compară greutatea căutată cu greutatea obiectului din mijloc (**m**), dacă valoarea căutată este la dreapta lui **m**, **p** devine **m** + 1; dacă valoarea căutată este la stânga lui **m**, **u** devine **m** - 1.

Secvența de valori în care se face căutarea este delimitată de adresele **p** și **u**. Căutarea are sens doar dacă $p \leq u$. Dacă se ajunge la situația $p > u$, valoarea căutată nu există în secvență și se afișează mesajul corespunzător.

Raționamentul și algoritmul descris nu tratează cazul particular în care valoarea căutată se află chiar pe prima sau ultima poziție din vector.

Pentru generalizare, se verifică întâi cazul particular. Dacă valoarea căutată nu se află pe prima sau pe ultima poziție, se intră în secvența căutărilor prin înjumătățire.

2. Reprezentarea algoritmului

început căutare_rapidă

citește A //valoare căutată

// aici pot fi introduse instrucțiunile

// pentru tratarea cazului particular

dacă $(A > G[1])$ și $(A < G[N])$

atunci

$p \leftarrow 1$

$u \leftarrow N$

$găsit \leftarrow 0$

repetă

$m \leftarrow (p+u) \text{ div } 2$

dacă $A = G[m]$

atunci

$găsit \leftarrow 1$

altfel

dacă $A > G[m]$

atunci

$p \leftarrow m+1$

altfel

$u \leftarrow m-1$

sf dacă

sf dacă

până când $(găsit = 1)$ sau $(p > u)$

dacă $găsit = 0$

atunci scrie "valoare inexistentă"

altfel scrie p

sf dacă

altfel

scrie "valoare inexistentă"

sf dacă

sfârșit cautare_rapidă

3. Tabelul de variație

Verificați algoritmul completând tabelul de variație pentru vectorul următor:

G	3	7	9	15	21	24	28	30	32	36	43	51	62	67	90
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

Tabel de variație pentru vectorul G

Tabelul 26

A	p	u	m	G [m]	Rezultatul căutării
67	1	15	8	30	
	9		12	51	$p \leftarrow m+1$
					$p \leftarrow m+1$
40	1	15	8	30	$p \leftarrow m+1$
	9		12	51	$u \leftarrow m-1$
		11	10	36	$p \leftarrow m+1$
	11		11	43	$u \leftarrow m-1$
		10			$p > u$
					Valoare inexistentă
15					
20					
2					

Metoda căutării rapide prin înjumătățiri succesive se numește **căutare binară** și folosește tehnica cuceritorului: **Divide et Impera**.

Divide et Impera

1. **Divide et Impera** este o tehnică de rezolvare a problemelor prin descompunere în probleme din ce în ce mai simple, până la o problemă cu soluție imediată.

2. **Căutarea binară** folosește metoda **Divide et Impera**. Problema inițială – căutarea unei valori într-un vector ordonat – este descompusă în probleme de același fel, dar din ce în ce mai simple, prin reducerea intervalului de valori în care se face căutarea până la un singur element.

După ce v-ați însușit elementele de bază ale proiectării algoritmilor, puteți „ataca” cu încredere orice limbaj de programare. Fiți pregătiți să învățați cât mai multe limbaje. Înarmați cu puterea bitului și cu o gândire corectă și rapidă, veți fi cuceritori.

Teste – răspunsuri

8. 1. b. 2. c. 3. a. 4. 8 și 3. 5. 1, 6 și 3. **10.** 1.1. b. 1.2. c. 1.3. a. 1.4. c. 2.1. b. 2.2. a. 3.1. orice pereche x, y de valori egale. 3.2. orice pereche de valori x, y cu $x < y$. 4. c. 5.1. $x > y$ și $z = 0$ sau $x = 1, y = 0$ și $z = 1$. 5.2. $x = 0, y < 0$. 5.3. $x < y$ și $z = 0$. 6. b și c. **13.** 1. d. 2.1. a oricare și $b = 0$. 2.2. b oricare și $a = 0$. 2.3. $a = 0, b = 0$. 3. a, b și c. 4. b. **14.** 1. b. 2.1. b. 2.2. $y < 10$. 3. a – F, b – F; c – A. 4. a. 5. b. 6. a – A, b – A, c – F. **18.** 1.1. b. și c. 1.2. $S = n(n + 1)/2$. 2.1. a și c. 3. c. 4. d. 5.1. 15. 5.2. $x > y$ (contor crescător). 5.3. $y(y + 1)/2 - x(x - 1)/2$. 6. b. **20.** 1. b. 2. c. 3. orice vector cu doar 3 elemente zero. 4. c. 5. b. 6. c. 7. b. 8. d.

ÎN LOC DE RECAPITULARE

- ✓ Calculatorul nu gândește în locul nostru.
- ✓ Calculatorul nu gândește cum vrem noi.
- ✓ Calculatorul gândește și execută așa cum îl învățăm noi.
- ✓ Și calculatorul învață!
- ✓ Profesorul calculatorului este programatorul.
- ✓ Cu rațiune și imaginație, programatorul construiește scenariul logic după care gândește calculatorul.
- ✓ Orice program modelează și comunică calculatorului un algoritm.
- ✓ Un algoritm corect structurat e ușor de codificat în limbaj de programare.
- ✓ Un algoritm scurt și simplu generează un program ușor de scris și de testat.
- ✓ Repetiția e mama algoritmilor.
- ✓ Structurile repetitive scurtează reprezentarea algoritmilor dar, dacă nu sunt corect construite, pot lungi timpul de execuție până la ... **time out!**
- ✓ Înainte de a învăța calculatorul să gândească, să învățăm noi înșine să gândim corect.
- ✓ Există metode și tehnici de gândire rapide.
- ✓ Imaginația și creativitatea de gândire sunt nelimitate.
- ✓ Deopotrivă știință și artă, programarea e o provocare continuă.
- ✓ Înarmat cu puterea bitului și cu o gândire corectă și rapidă, programatorul este un cuceritor.

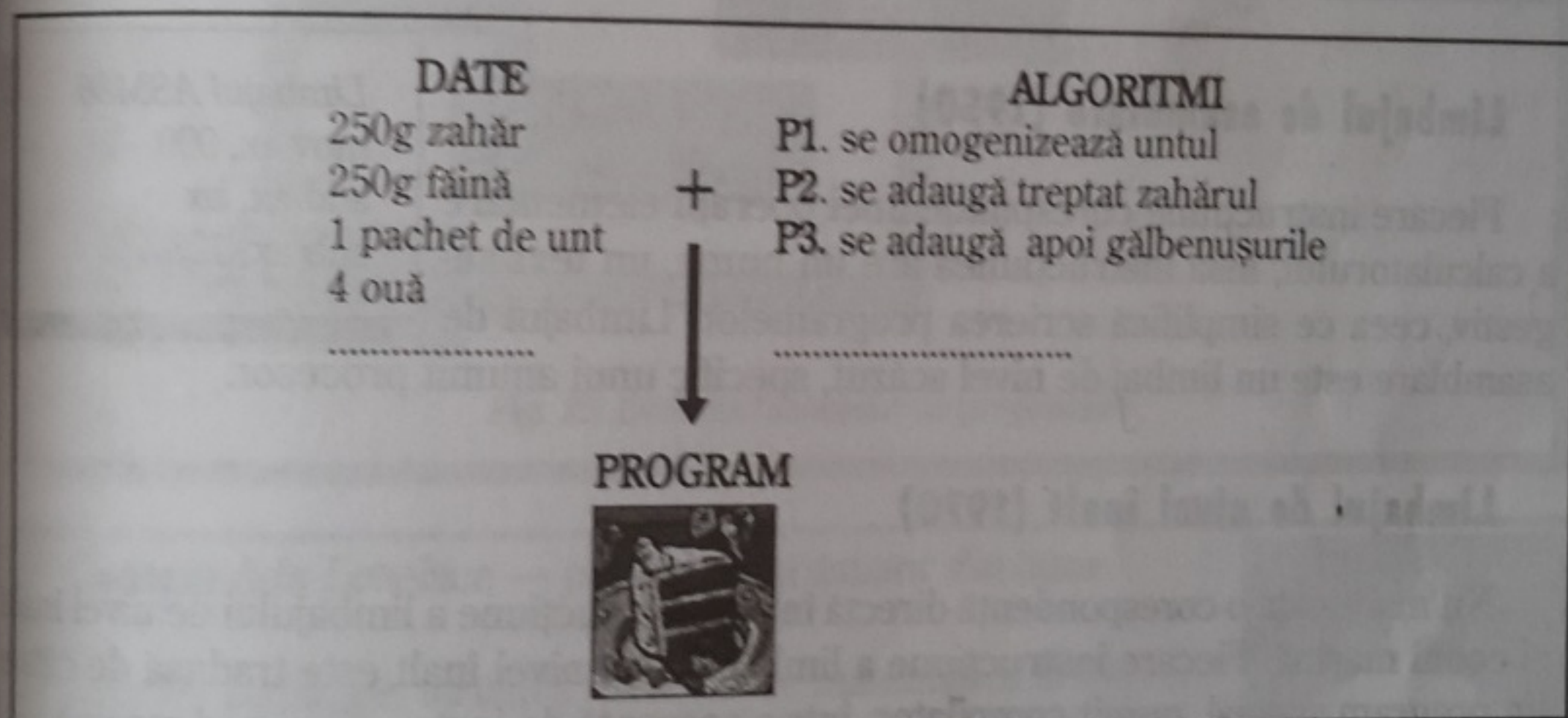
PARTEA A II-A

ARTA DE A PROGRAMA

Un program poate fi asemenea unui obiect de artă,
în fața căruia se poate exclama:
„Ce soluție frumoasă!”

1. Limbaje programare

Un program scris în orice limbaj de programare este asemenea unei rețete de preparare a unei prăjituri.



Limbajul de programare lucrează cu cele două concepte prezentate în partea I: **date** și **algoritmi**.

Datele reprezintă informația procesată de program, iar *algoritmii* reprezintă metodele utilizate de program.

Limbajul de programare este un limbaj artificial care asigură comunicarea între programator și calculator.

Învățarea unui limbaj de programare se aseamănă cu învățarea unei limbi străine, limbajul de programare având și el un vocabular (mult mai restrâns decât al oricărei limbi străine), un set de reguli sintactice și semantice, care trebuie respectate la formarea propozițiilor.

Propozițiile scrise într-un limbaj de programare se numesc **instrucțiuni**.

Programul sursă este o listă de instrucțiuni prin care programatorul indică modul în care calculatorul trebuie să prelucreze informația primită.

Istoria limbajelor de programare

Limbajul mașină

Limbajul mașină, limbajul „nativ” al calculatorului, este o secvență de coduri binare, adică un șir de 0 și 1.

Între anii 1940-1950 programatorii scriau toate programele folosind numai cifre binare, 0 și 1, cunoscând foarte bine modul în care calculatorul interpreta diferitele combinații de 0 și 1.

Programele devenind tot mai mari, acest mod de a programa era tot mai incomod. Au fost create limbaje de programare care permiteau exprimarea unei instrucțiuni într-o formă mai accesibilă omului.



Fig. 23. ENIAC – primul calculator electronic

Limbajul de asamblare (1950)

Fiecare instrucțiune corespunde unei operații elementare a calculatorului, însă instrucțiunea are un nume, un text sugestiv, ceea ce simplifică scrierea programelor. Limbajul de asamblare este un limbaj de nivel scăzut, specific unui anumit procesor.

Limbajul ASM86
mov ax, 999
add ax, bx
sub ax, 123

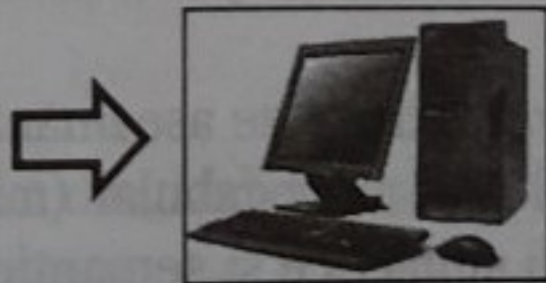
Limbajul de nivel înalt (1970)

Nu mai există o corespondență directă între o instrucțiune a limbajului de nivel înalt și codul mașină. Fiecare instrucțiune a limbajului de nivel înalt este tradusă de către un program special, numit **compilator**, într-o secvență de instrucțiuni cod-mașină.

Limbajul de nivel înalt este mult mai apropiat de limbajul natural și de raționamentul uman.

```
program test;  
var raspuns:integer;  
begin  
  write('7x9=?');  
  readln(raspuns);  
  if raspuns=63 then  
    write('FB')  
  else  
    write ('Mai învață!');  
  readln;  
end.
```

Program sursă Pascal
nume_fisier.pas



Compilator

```
0001001101111111  
0011001100010100  
1100110000110001  
1111001110001110  
0011000011110000  
1000001101010101  
0100101010101011  
0110101011111111  
0101010111111010  
0101101001011111  
1000011111111100
```

Program executabil
nume_fisier.exe

Fig. 24. Traducerea din limbajul de nivel înalt în cod mașină

Un limbaj de nivel înalt nu mai depinde de hardware. Astfel, se poate utiliza același program scris într-un limbaj de nivel înalt pe diferite platforme, utilizând compilatoare separate pentru fiecare platformă.

Limbajul natural vorbit

Cercetări pentru recunoașterea și înțelegerea vorbirii sunt efectuate, în prezent, în multe laboratoare de cercetare. Este un proiect foarte important și deloc ușor de realizat.

De exemplu, sistemul de rezervări aeriene Pegasus poate interpreta câteva mii de cuvinte care pot fi vorbite și conectate în propoziții.

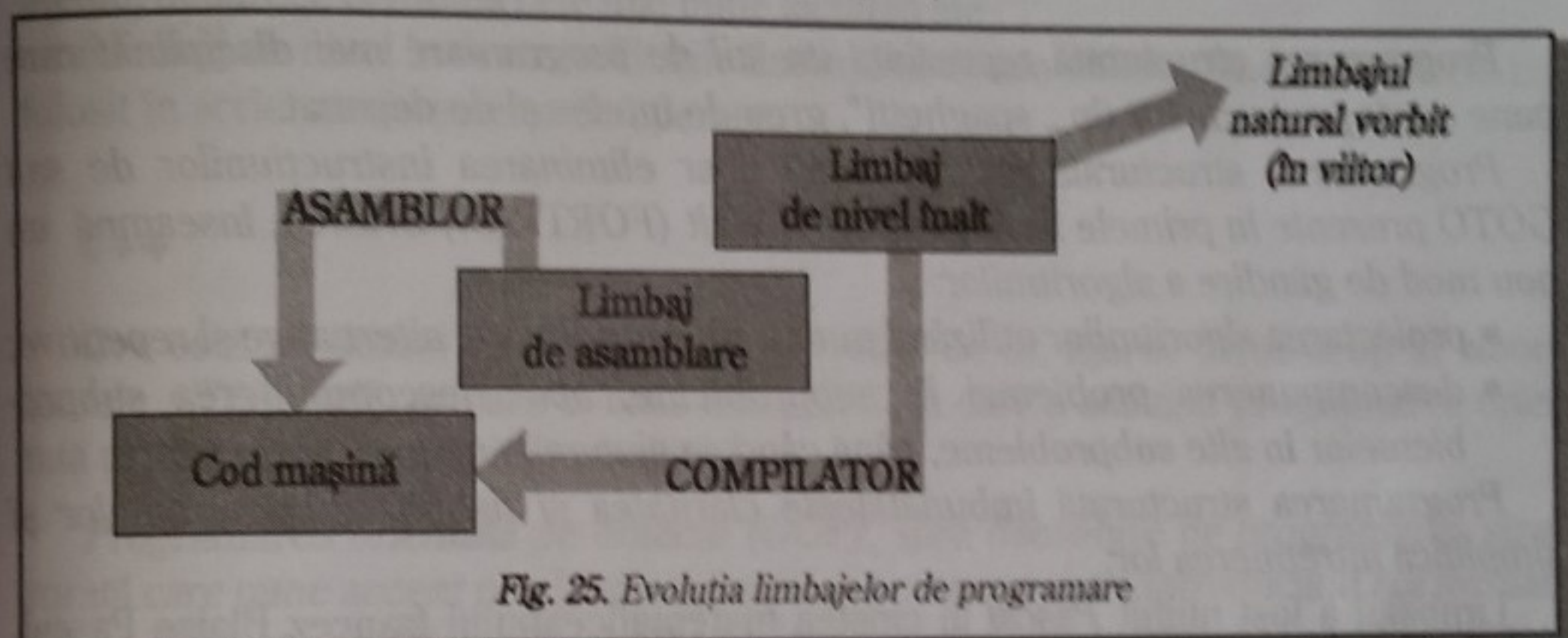


Fig. 25. Evoluția limbajelor de programare

Contesa Ada Lovelace — primul programator din lume

Charles Babbage, strălucit profesor de matematică la Universitatea Cambridge, a proiectat în anul 1834 prima mașină mecanică de calcul care funcționa pe baza unor principii care stau la baza calculatoarelor electronice moderne.

Contesa Ada Lovelace, fiica poetului Lord Byron, a descris operațiile necesare pentru calculul numerelor Bernoulli, utilizând mașina proiectată de Charles Babbage, scriind astfel primul program din istoria calculatoarelor.

Limbajul de programare ADA este un omagiu adus contesei Ada Lovelace, supranumită „magiciana numerelor”.



Fig. 26

Contesa Ada Lovelace

Limbaje de nivel înalt

Basic

Creat în anul 1964 la Dartmouth College (SUA), limbajul Basic era ușor de învățat și disponibil pe toate tipurile de calculatoare. Dacă inițial Basic-ul era un limbaj

nestructurat, ulterior au apărut versiuni care au introdus elemente de programare structurată. Limbajul Basic, popular datorită simplității sale, a avut succes într-o perioadă în care aplicațiile dezvoltate nu erau foarte pretențioase.

Pascal

Niklaus Wirth, profesor la Universitatea din Zürich, a creat în anul 1970 un limbaj prin care a introdus un nou stil de programare, **programarea structurată**. Versiunea inițială a limbajului a fost destinată predării programării calculatoarelor în cadrul universității.

Programarea structurată reprezintă un stil de programare mai disciplinat care pune capăt programelor tip „spaghetti”, greu de înțeles și de depanat.

Programarea structurată nu înseamnă doar eliminarea instrucțiunilor de salt GOTO prezente în primele limbaje de nivel înalt (FORTRAN, BASIC), înseamnă un nou mod de gândire a algoritmilor:

- proiectarea algoritmilor utilizând numai structuri liniare, alternative și repetitive;
- descompunerea problemei în subprobleme, apoi descompunerea subproblemelor în alte subprobleme, până când se ajunge la operații elementare.

Programarea structurată îmbunătățește claritatea și fiabilitatea programelor și simplifică întreținerea lor.

Limbajul a fost numit **Pascal** în cinstea matematicianului francez Blaise Pascal, care a construit, în anul 1642, primul dispozitiv de numărare.

Limbajul Pascal, în ciuda complexității sale, nu este dificil de învățat, datorită clarității ireproșabile. Despre limbajul Pascal se poate spune: „*fiecare lucru la locul potrivit*”.

Calitățile sale au făcut ca, foarte curând, limbajul Pascal să treacă dincolo de zidurile universității și să cucerească lumea.

Implementarea cea mai spectaculoasă a acestui limbaj a fost **Turbo Pascal**, implementare realizată de firma BORLAND. **Turbo Pascal** a revoluționat domeniul programării calculatoarelor prin performanțele compilatorului și prin ușurința de utilizare a mediului integrat de programare.

Versiunile standard ale implementărilor Pascal sunt: **Turbo Pascal 6.0**, **Borland Pascal 7.0** (include programarea orientată pe obiecte), **Borland Delphi**.

Claritatea, regulile sintactice stricte, ușurința depanării, precum și geneza sa didactică, îl recomandă ca prim limbaj de programare.

Studiile efectuate în rândul programatorilor au stabilit că persoanele pentru care limbajul Pascal a fost primul limbaj de programare învățat, învață mult mai ușor noi limbaje de programare.

Acestea sunt motivele pentru care am ales limbajul Pascal pentru realizarea acestui ghid practic de programare.

Versiunea utilizată în scrierea programelor este **Borland Pascal 7.0**. Puteți utiliza, fără probleme, versiunea **Turbo Pascal 6.0**.

C

A fost creat în anul 1972 de către Dennis Ritchie și Brian Kernigham pentru dezvoltarea sistemului de operare UNIX, un sistem de operare universal, care să poată funcționa, teoretic, pe orice tip de mașină.

Calitățile esențiale ale limbajului sunt un cod eficient pentru operațiile de intrare/ieșire și portabilitatea programelor scrise în C.

Portabilitatea este proprietatea unui software scris pe un anumit tip de calculator și un anumit sistem de operare de a se adapta ușor pe alt sistem de operare sau alt sistem de calcul.

Limbajul C este considerat un limbaj de nivel mediu, deoarece combină cele mai bune facilități ale unui limbaj de nivel înalt cu facilitățile limbajului de asamblare, fiind limbajul cel mai folosit în scrierea sistemelor de operare.

C++

C++ este un limbaj de programare dezvoltat de dr. Bjarne Stroustrup în laboratoarele AT&T Bell care are la bază limbajul C, la care a adăugat programarea orientată pe obiecte precum și alte facilități.

Programarea orientată pe obiecte (OOP), spre deosebire de programarea structurată care pune accent pe algoritmi, va pune accent pe date, introducând noi formate. **Clasa** definește ce date sunt folosite pentru a reprezenta un obiect și ce operații pot fi executate asupra acestor date.

OOP facilitează scrierea de cod reutilizabil, ușurând mult munca programatorilor. Producătorii compilatoarelor pun la dispoziția programatorilor o mulțime de biblioteci de clase utile dezvoltării aplicațiilor sub sistemul de operare Windows.

Limbajele de nivel înalt, **C**, **C++** și **Pascal** sunt cele mai puternice limbaje profesionale.

2. Limbajul de programare PASCAL

Vocabularul limbajului Pascal

Un limbaj de programare, la fel ca limbajul natural, este definit prin vocabular, sintaxă și semantică.

1. Vocabularul

- alfabet:

- literele mari și mici ale alfabetului englez (a..y, A..Y), cifrele (0..9), caractere speciale (+ - * / = () [] { } ^ < > , . : ; # \$ @ ' spațiu);

2. **blancuri:** caracterul spațiu, caracterele de control (codurile ASCII de la 0 la 31).
- **cuvinte:**
 - a) **identificatori:** o succesiune de litere, cifre și caracterul `_` (caracterul de subliniere), care poate începe cu o literă; sunt semnificative doar primele 63 de caractere ale unui identificator; nu se face distincție între litere mici și mari;
 - b) **cuvinte rezervate (cheie):** identificatori care nu pot fi folosiți în alt context decât cel specificat; cuvintele rezervate se folosesc, în general, la scrierea instrucțiunilor și nu pot fi redefinite;
 - c) **comentarii:** comentariile sunt șiruri de caractere delimitate de caracterele `(* *)` sau `{ }` care sunt ignorate de către compilator.

Lista celor mai utilizate cuvinte cheie ale limbajului Pascal

Tabelul 27

and	else	in	program	then
array	end	label	record	until
begin	file	mod	repeat	var
case	for	nil	set	while
const	function	not	then	with
div	forward	of	to	xor
do	goto	or	string	
downto	if	procedure	text	

2. **Sintaxa:** totalitatea regulilor de formare a propozițiilor, adică a instrucțiunilor. Nerespectarea acestor reguli de către programator conduce la erori de sintaxă, care fac imposibilă compilarea programului.
3. **Semantica:** totalitatea regulilor prin care se asociază semnificații instrucțiunilor.

3. Copii celebri din lumea calculatoarelor

Copiii celebri din lumea calculatoarelor au scris primele programe în timpul gimnaziului și au devenit inventatori în anii studenției, revoluționând lumea calculatoarelor.

◆ **Steve Wozniak și Steve Jobs**, un student și un proaspăt absolvent al universității, inventatorii primului calculator personal, **APPLE** (anul 1975).

În timpul școlii, Steve Wozniak a avut mulți prieteni pasionați de electronică. Jucându-se, și-au construit propriile aparate de comunicație. „Este plăcut să crești într-un grup de copii pasionați de electronică (electronics kids)”, afirma inventatorul primului calculator personal.

◆ **Bill Gates** – o legenda vie a lumii IT

Numele lui Bill Gates este legat, în primul rând, de compania Microsoft, de sistemul de operare Windows.



Fig. 27.
Steve Wozniak și Steve Jobs

Elev fiind la Lakeside School, a scris primul program Basic la vârsta de 13 ani. Împreună cu alți trei prieteni a format un grup hotărât să descopere secretele computerelor, pe care l-au numit Lakeside Programming Group.

În anii studenției, Bill Gates și Paul Allen (cofondatorul firmei Microsoft), au scris celebrul interpretor **BASIC**, făcând din Basic un limbaj foarte popular.

Acum, la numai 48 de ani, Bill Gates este o legendă vie a lumii calculatoarelor.



Fig. 28.
Bill Gates

◆ **Linus Torvalds** – autorul sistemului de operare Linux

În anul 1991, Linus Torvalds, student la Universitatea din Helsinki, a scris nucleul sistemului de operare Linux și l-a distribuit gratuit pe Internet.

În anul 1981, Linus era un băiat pasionat de programare. Primele programe scrise în limbajul Basic au fost niște jocuri realizate pe un calculator Commodore, un cadou de la bunicul său.

A scris apoi, în limbaj mașină, un joc video cu submarine, pe care l-a înregistrat pe o casetă și l-a dus la școală colegilor săi.



Fig. 29.
Linus Torvalds

4. În culisele limbajului de programare

Mașina „cu două degete”

Pentru reprezentarea tuturor informațiilor se folosesc numere, mașinile de calcul având nevoie numai de două cifre, 0 și 1, pentru memorarea oricărui număr.

Oamenii folosesc 10 cifre (0..9) pentru reprezentarea numerelor, deoarece cele 10 degete de la mâini au constituit prima socotitoare a omului.

În timp ce omul numără 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10... calculatorul, „mașină cu două degete”, numără 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010...

0 sau 1 se numește **bit** (**b**inary **d**igit) și reprezintă cea mai mică unitate de măsură a informației. Bit-ul este „grăuntele de nisip” cu ajutorul căruia sunt construite structuri de informație mult mai mari.

O colecție de 8 biți, considerați ca o singură unitate, se numește **byte** sau **octet**.

Dacă bit-ul este „grăuntele de nisip” al informației, byte-ul este „cărămida” blocurilor uriașe de informații.

O succesiune de 1024 bytes formează un **kilobyte**.

1 kilobyte = 1024 bytes = 2^{10} Bytes; 1 megabyte = 1024 kB = 2^{20} Bytes;

1 gigabyte = 1024 MB = 2^{30} Bytes; 1 terabyte = 1024 GB = 2^{40} Bytes.

Calculatorul este format din milioane de tranzistoare care acționează ca niște comutatoare: deschise fiind, nu permit trecerea curentului (0), închise fiind, permit trecerea curentului (1). Cifrele binare 1 sau 0 reprezintă, de fapt, semnale electrice produse în interiorul calculatorului și anume, atunci când într-o locație este prezentă o anumită tensiune, se spune că acea locație conține valoarea 1, când nu există tensiune, acea locație este interpretată ca având valoarea 0.

Comutatoarele sunt organizate în grupe de câte 8, numite bytes sau octeți. Cu ajutorul unui număr binar format din 8 cifre binare se pot reprezenta intern toate caracterele de pe tastatură.

Reprezentarea internă a informației nenumерice

Pentru reprezentarea tuturor informațiilor, calculatorul folosește numere scrise numai cu cifrele 0 și 1. Cum va reprezenta atunci litera **A**?

Printr-un cod numeric format cu ajutorul a 8 biți, și anume 01000001.

Litera **B** va fi reprezentată prin șirul de cifre binare 01000010, iar litera **a** prin șirul 01100001.

Pentru ca un text codificat pe un calculator să poată fi decodificat corect pe un calculator din altă țară s-a creat un sistem standard de codificare a informației alfanumerice, numit **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange).

Codurile ASCII sunt prezentate în anexa de la finele cărții.

Sisteme de numerație

Oamenii au știut să socotească înainte de a ști să scrie, folosind degetele, însemnările făcute pe pereții peșterilor, oasele, funia cu noduri, pietricele, abacul etc.

Sistemul de numerație reprezintă totalitatea regulilor de reprezentare a numerelor folosind o mulțime de simboluri, numite **cifre**.

Baza unui sistem de numerație este dată de numărul maxim de cifre utilizat în scrierea numerelor în acel sistem de numerație.

Primele sisteme de numerație au apărut în urmă cu 4000 de ani. Cele mai cunoscute au fost sistemul roman, chinezesc, grecesc, arabic (strămoșul sistemului din zilele noastre).

Sisteme de numerație poziționale

Sistemele de numerație zecimal, binar, hexazecimal sunt sisteme de numerație poziționale, în care fiecare cifră a numărului are un anumit rang în cadrul numărului (tabelul 30). Cifrele unui număr nu diferă doar prin valoare, ci și prin poziția lor în cadrul numărului.

Exemplu:

$$9713_{(10)} = 3 + 10 + 700 + 9000 \quad \text{altfel scris: } 9713_{(10)} = 9 \cdot 10^3 + 7 \cdot 10^2 + 1 \cdot 10^1 + 3 \cdot 10^0$$

Tabelul 28

Sistemul de numerație	Baza	Mulțimea cifrelor	Utilizare
Zecimal	10	{0; 1; 2; 3; 4; 5; 6; 7; 8; 9}	activități umane
Binar	2	{0; 1}	circuite electronice
Hexazecimal	16	{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; A; B; C; D; E; F} unde A = 10; B = 11; C = 12; D = 13; E = 14; F = 15	programarea calculatoarelor

În general, numărul $x = \overline{a_n a_{n-1} \dots a_1 a_0}$ se va scrie în baza b astfel:

$$x = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b^1 + a_0 b^0, \text{ unde } b \text{ este un număr natural, } b > 1.$$

Numărul maxim de n cifre care poate fi reprezentat în baza b este $b^n - 1$ (tabelul 31).

Exemple:

- numărul maxim de 4 cifre care se poate reprezenta în baza 10, este 9999, adică $10^4 - 1$.

- numărul maxim de 4 cifre care se poate reprezenta în baza 2, este $1111_{(2)}$.

$$(1111)_{(2)} = 15_{(10)}, \text{ iar } 15 = 2^4 - 1.$$

Tabelul 29

Nr. maxim de cifre	Numărul maxim în baza 2	Nr. maxim în baza 10	Nr. maxim = $2^n - 1$
4	1111	15	$15 = 2^4 - 1$
8	11111111	255	$255 = 2^8 - 1$
16	1111111111111111	65535	$65535 = 2^{16} - 1$
32	11111111111111111111111111111111	2147483647	$2147483647 = 2^{32} - 1$

Conversia dintr-o bază de numerație în alta

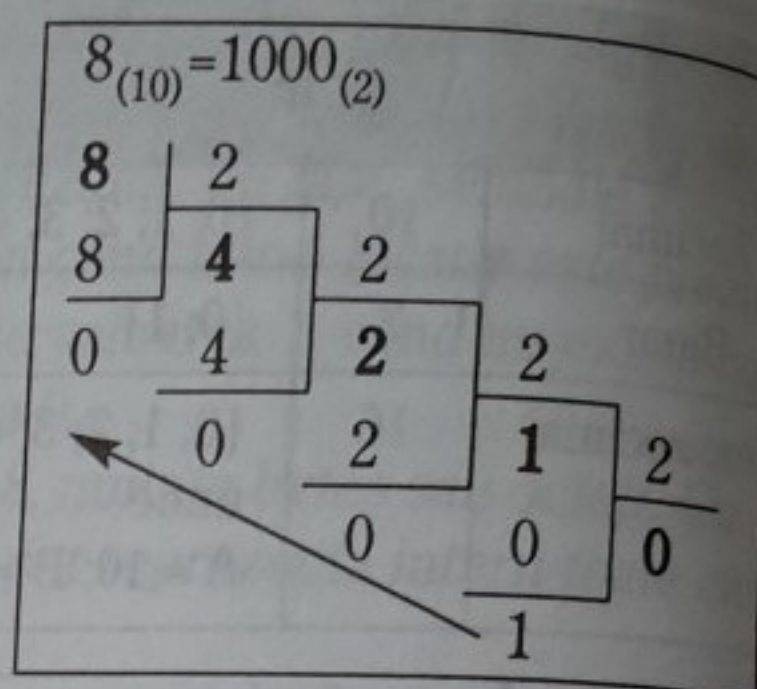
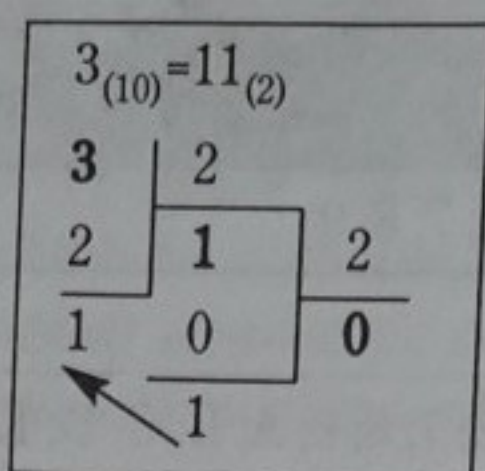
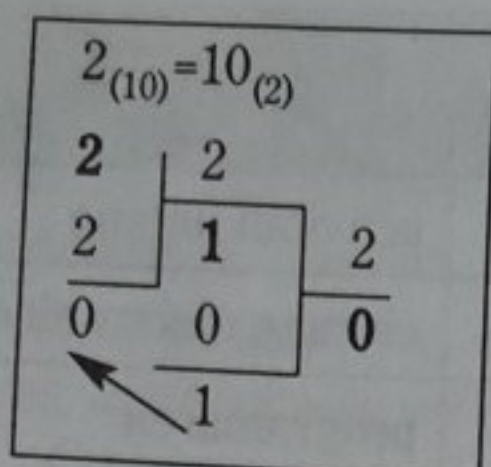
Conversia unui număr natural din baza 10 în baza 2

Când introducem de la tastatură datele care trebuie să fie prelucrate de către calculator, le introducem în baza 10. La intrarea în sistemul de calcul datele sunt convertite în baza 2, adică sub forma unor secvențe de 0 și 1.

Conversia unui număr din baza 10 în baza 2 se face prin împărțiri succesive la baza în care convertim, adică la 2, până când câtul devine 0.

Resturile obținute la aceste împărțiri succesive, începând de la ultimul rest și până la primul rest, formează scrierea binară a numărului zecimal.

Exemple:



$$8_{(10)} = 1000_{(2)}$$

↑ cifra cea mai puțin semnificativă

Observații

1. Prima cifră binară care se obține cu algoritmul de mai sus reprezintă cea mai puțin semnificativă cifră a numărului, iar ultima cifră care se obține este cea mai semnificativă cifră a numărului scris în baza 2.
2. Algoritmul de mai sus este valabil pentru orice altă bază b , b fiind un număr natural, $b > 1$.

Scrierea unor numere în trei baze diferite de numerație: 10, 2, 16

Tabelul 30

Baza 10	Baza 2	Baza 16
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A

Baza 10	Baza 2	Baza 16
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F
16	00010000	10
32	00100000	20
64	01000000	40
127	01111111	7F
128	10000000	80
255	11111111	FF

Observații

1. Fiecare număr zecimal s-a reprezentat în baza 2 pe 8 biți (1 byte), byte-ul fiind cea mai mică locație pentru memorarea datelor. Biții se numerează începând cu 0, astfel încât un byte este format din 8 biți numerotați de la 0 la 7.
2. La fel ca în baza 10, și în baza 2 putem adăuga în fața numărului câte zerouri dorim, fără să modificăm valoarea numărului.

3. Fiecare grup format din 4 biți (de la dreapta la stânga) se reprezintă printr-o singură cifră hexazecimală.

Conversia unui număr natural din baza 2 în baza 10

Exemple: • Reprezentarea internă a numărului $8_{(10)}$

Puterile lui 2	→	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
		0	0	0	0	1	0	0	0
Poziția biților în cadrul numărului	→	7	6	5	4	3	2	1	0

$$1000_{(2)} = 0 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 0 + 0 + 0 + 8 = 8_{(10)}$$

$$\bullet 10_{(2)} = 0 \cdot 2^0 + 1 \cdot 2^1 = 0 + 2 = 2_{(10)}$$

$$\bullet 11_{(2)} = 1 \cdot 2^0 + 1 \cdot 2^1 = 1 + 2 = 3_{(10)}$$

Conversia în baza 10 a unui număr scris în baza 2 se obține însumând toate produsele parțiale obținute prin înmulțirea fiecărei cifre binare a numărului cu 2^i , unde i este poziția cifrei în cadrul numărului binar.

Conversia unui număr subunitar din baza 10 în baza 2

Exemplu: $0,875_{(10)} = 0,1110_{(2)}$

Se înmulțește partea fracționară a numărului cu 2 și se separă partea întreagă și partea fracționară ale produsului obținut. Partea întreagă reprezintă o cifră binară a numărului în baza 2, iar partea fracționară se înmulțește în continuare cu 2.

Procedeul se reia până când se obține produsul 0 sau numărul de cifre dorit.

$$\begin{aligned} 0,875 \cdot 2 &= 1,750 = 0,750 + 1 \\ 0,750 \cdot 2 &= 1,500 = 0,500 + 1 \\ 0,500 \cdot 2 &= 1,0 = 0,0 + 1 \\ 0,0 \cdot 2 &= 0 \end{aligned}$$

Operații aritmetice cu numere scrise în baza 2

Operațiile aritmetice în baza 2 se derulează după aceleași reguli ca operațiile aritmetice în baza 10.

Adunarea a două numere binare

Baza 10	Baza 2
$\begin{array}{r} 16+ \\ \underline{16} \\ 32 \end{array}$ 6 + 6 = 12 Se scrie cifra unităților = 2. Cifra zecilor = 1 reprezintă transportul către rangul următor	$\begin{array}{r} 00010000+ \\ \underline{00010000} \\ 00100000 \end{array}$ 1+ <u>1</u> 10 ($2_{(10)}$)

La adunarea a două cifre binare de același rang pot apărea următoarele cazuri:

$$\begin{array}{r} 0+ \\ \underline{0} \\ 0 \end{array} \quad \begin{array}{r} 0+ \\ \underline{1} \\ 1 \end{array} \quad \begin{array}{r} 1+ \\ \underline{1} \\ 10_{(2)} \rightarrow 2_{(10)} \end{array}$$

Scăderea a două numere binare

Baza 10	Baza 2
$\begin{array}{r} 14- \\ \underline{5} \\ 9 \end{array}$ 4 < 5 $\Rightarrow$ se face un împrumut de la rangul următor de valoare egală cu baza, adică 10. Împrumutul se va adăuga la 4: 4 + 10 = 14 $\Rightarrow$ 14 - 5 = 9 La ordinul zecilor se va restitui împrumutul: 1 - 1 = 0	0 < 1 $\Rightarrow$ se va face un împrumut de la rangul următor $\begin{array}{r} 00001110- \\ \underline{00000101} \\ 00001001 \end{array}$ 10 - <u>1</u> 1 Valoarea împrumutului este $10_{(2)} = 2_{(10)}$

La scăderea a două cifre binare de același rang pot apărea următoarele cazuri:

$$\begin{array}{r} 0- \\ \underline{0} \\ 0 \end{array} \quad \begin{array}{r} 1- \\ \underline{1} \\ 0 \end{array} \quad \begin{array}{r} 0- \\ \underline{1} \\ 1 \end{array}$$

Se va face un împrumut în valoare de $10_{(2)} = 2_{(10)}$
 $10_{(2)} - 1_{(2)} = 1_{(2)}$

Probleme propuse

1. Prin analogie cu înmulțirea a două numere zecimale, încercați să înmulțiți următoarele numere binare:

a) 1001^* b) 11010^*

$$\begin{array}{r} 101 \\ \underline{1010} \end{array}$$

5. Memoria ca o tablă de scris

Memoria internă. Variabile de memorie

Informațiile prelucrate de calculator (programe și valori) se păstrează în memoria internă a calculatorului, numită memorie **RAM** (**R**andom **A**ccess **M**emory) care are următoarele caracteristici:

- este formată din mai multe circuite electronice;
- permite acces la citire și la scriere;
- este volatilă — după execuția programului, memoria alocată este eliberată.

Memoria internă RAM este centrul activității calculatorului, este asemenea unei table dintr-o sală de clasă, pe care procesorul „scrie” și „șterge” date.

Organizarea memoriei interne

Memoria internă este organizată sub forma unor locații succesive, de câte 1 byte. Pentru a putea extrage date din memorie sau pentru a scrie date în memorie trebuie să cunoaștem adresa de unde extragem sau unde memorăm datele.

Astfel, fiecare locație de 1 byte are un număr de ordine în cadrul memoriei, numit *adresă de memorie*.

Câte 65 536 de locații de câte 1 byte sunt grupate într-un segment de memorie cu dimensiunea de 64 kB ($65\,536 \text{ bytes} : 1024 = 64 \text{ kB}$; 1 kB având 1024 bytes). Locațiile din care este format segmentul sunt numerotate de la 0 la 65 535. Acest număr reprezintă deplasamentul (offset-ul) locației în cadrul segmentului din care face parte.

Ca urmare, adresa unei locații de memorie se compune din două câmpuri: adresa segment și offset (deplasament) (tabelul 31).

Tabelul 31

Adresa fizică de memorie		
adresa segment	:	offset (deplasament)
numărul de ordine al segmentului din care face parte locația de memorie	:	numărul de ordine al locației în cadrul segmentului

Exemple:

- 0000: 0000 (prima locație din memoria RAM)
- 0000: 046C (adresa locației pentru contorizarea timpului de execuție a unui program)

Utilizarea adreselor fizice de memorie în scrierea programelor era foarte incomodă, astfel încât, în limbajele de nivel înalt, adresele fizice de memorie sunt înlocuite cu identificatori, nume simbolice, ușor de utilizat de către programatori. Compilatorul păstrează în niște tabele interne corespondența dintre numele simbolice și adresele fizice de memorie ale locațiilor utilizate.

Variabila de memorie

Variabila reprezintă un loc în memoria internă a calculatorului (figura 28). Conținutul variabilei se poate modifica pe parcursul execuției programului, de aceea se numește *variabilă*. Noua valoare se va „suprascrie” peste vechea valoare, care se va „pierde”.

Adresa fizică	Memoria internă	Numele variabilei
0000:046C	variabilă	time
0000:0000		

Fig. 30. Reprezentarea variabilei în memoria calculatorului

Într-o variabilă se poate memora o **valoare** de un anumit **tip**.

Harta memoriei

La lansarea în execuție a unui program Pascal, codul mașină al programului se încarcă în memoria internă a calculatorului, ocupând mai multe segmente. Pentru memorarea datelor pe care le prelucrează programul se folosește un singur segment de date, de **64 KB**, adică 65 536 locații de câte 1 byte. **Segmentul de date** conține toate constantele cu tip și variabilele globale.

adrese mari	lista blocurilor libere de memorie
	↑ HEAP variabile dinamice
	↓ STIVA
	Segment de date • variabile globale • constante cu tip
	Instrucțiuni Pascal • cod unit System • cod prima unitate • • cod ultima unitate • cod program principal
0000:0000	Prefixul programului Pascal
	Informații sistem

Fig. 31. Alocarea memoriei în Turbo Pascal

6. Tipuri predefinite de date

Valorile memorate în variabile pot fi numere întregi sau reale, caractere sau șiruri de caractere.

Prin **tipul unei variabile** se definesc următoarele :

- mulțimea valorilor posibile;
- mulțimea operațiilor definite pe mulțimea valorilor;
- modul de reprezentare internă a valorilor.

Tipuri predefinite de date

Tabelul 32

Tipuri simple de date	Domeniul de valori	Reprezentarea datelor		Operatori	Funcții standard
		Spațiul de memorie	Reprezentarea internă		
TIPURI ÎNTREGI					
BYTE	număr fără semn 0 .. 255	1 Byte	0 → 00000000 255→11111111	Aritmetici: + - * div mod	sqr(3) = 9 succ(5) = 6 pred(5) = 4
WORD	număr fără semn 0 .. 65535	2 Bytes	65535 → 1111111111111111	Relaționali: = < > <= >=	
SHORTINT	număr cu semn -128 .. +127	1 Byte	-128→10000000 +127→01111111 Bitul cel mai semnificativ este bitul de semn : 0 →numere pozitive 1→ numere negative	Logici: not and or xor	abs(-3) = 3
INTEGER	număr cu semn -32768 .. +32767	2 Bytes	-32768 → 10000000 00000000 +32767 → 01111111 11111111		
LONGINT	număr cu semn -2147483648+2147483647	4 Bytes	100000000000000000 0000000000000000 .. 011111111111111111 1111111111111111		
TIPURI REALE					
REAL	2.9E-39 .. 1.7E38	6 Bytes	Precizia: 11 cifre	Aritmetici: + - * /	abs(-1.3) = 1.3 sqrt(9) = 3
SINGLE	1.5E-45 .. 3.4E38	4 Bytes	Precizia: 7 cifre	Relaționali Logici	sqrt(2) = 1.414 2135624

Tabelul 32 (continuare)

DOUBLE	5.0E-324 .. 1.7E308	8 Bytes	Precizia: 15 cifre		(reprezentare aproximativă)
EXTENDED	3.4E-4932 .. 1.1E4932	10 Bytes	Precizia: 19 cifre		trunc(1.5) = 1 trunc(-1.5) = -1 round(1.5) = 2
COMP (un format spe- cial de întreg zecimal)	numerele întregi din intervalul: $-2^{63}+1 .. 2^{63}-1$	8 Bytes	Precizia: 19 cifre		round(-1.5) = -2 round(1.4) = 1 round(-1.4) = -1 int(1.5) = 1 int(-1.5) = -1 int(1.4) = 1 frac(1.4) = 0,4 int(-1.4) = -1
TIPUL LOGIC					
BOOLEAN	FALSE, TRUE	1 Byte	FALSE → 0 TRUE → 1	Relaționali Logici	false < true; true and true = = true; true and false = = false; true or false = = true
CHAR	caracterele ASCII	1 Byte	'A' → 65 '0' → 48	Relaționali Logici	'A' < 'B'; succ('A') = 'B' pred('B') = 'A' ord('A') = 65 chr(65) = 'A'

Reprezentarea în virgulă mobilă a numerelor reale

Numerele reale sunt normalizate, adică sunt înmulțite sau împărțite la puteri ale lui 2, prin care virgula este mutată în așa fel încât partea întreagă să fie 0, iar prima cifră a părții fracționare să fie diferită de 0.

Exemplu: $9,875_{(10)} = 1001,1110_{(2)}$. Numărul normalizat va fi $0,10011110 \times 10^{100}_{(2)}$, unde $10^{100}_{(2)} = 2^4_{(10)}$ (s-a mutat virgula spre stânga cu 4 poziții).

bit semn	caracteristica (cum se mută virgula)	partea fracționară (mantisa)
-------------	---	------------------------------

↓
 DOMENIUL de VALORI PRECIZIA — numărul de cifre semnificative

- **Precizia** unei valori definește gradul de exactitate al reprezentării acesteia.
- Datorită numărului fix de cifre binare alocat memorării numărului real reprezentarea unui număr cu multe cifre va fi **aproximativă**.

NOTAȚIA ȘTIINȚIFICĂ: $1,7E38 = 1,7 \times 10^{38}$

Completări

- Tipurile byte, word, integer, longint, char și boolean sunt:
 - tipuri simple*: păstrează o singură valoare la un moment dat;
 - predefinite*: sunt definite în unit-ul System, nu mai necesită declarare în secțiunea TYPE;
 - ordinale*: sunt definite funcțiile standard ca ord, succ, pred.
- Tipurile reale sunt tipuri simple predefinite.
- Pentru utilizarea tipurilor *double*, *extended* și *comp* este necesar coprocessorul matematic, partenerul procesorului în efectuarea calculelor în virgulă mobilă. Se poate utiliza directiva de compilare {\$N+} sau se poate selecta meniul Options → Compiler → Numeric processing → [x] 8087/80287.

7. Structura unui program Pascal

program nume_program;

const

declarații de constante;

type

declarații de tipuri utilizator

var

declarații de variabile;

Partea declarativă

begin

lista instrucțiunilor

end.

Partea executabilă

Declararea și inițializarea variabilelor

Orice variabilă este caracterizată de:

- adresa fizică de memorie*: gestionată de compilator;
- tipul variabilei*: declarat de către programator în secțiunea **var**;
- valoarea*: se introduce în variabilă printr-o instrucțiune de atribuire sau printr-o instrucțiune de citire.

program primul_program;

var

a: char;

b: char;

c: integer;

d: byte;

La citirea secțiunii **var** compilatorul alocă în segmentul de date spațiul de memorie necesar fiecărei variabile, conform tipului declarat.

Nu se introduce nici o valoare în spațiul alocat.

Segmentul de date	Numele variabilei	Spațiul de memorie alocat
	d	1 Byte
	c	2 Bytes
	b	1 Byte
	a	1 Byte

begin { Inițializarea variabilelor prin instrucțiuni de atribuire }

d:=3;		00000011	d	1 Byte
c:=-3;		11111101		
		11111111	c	2 Bytes
b:='A';	codul ASCII = 65	01000001	b	1 Byte
a:='3';	codul ASCII = 51	00110011	a	1 Byte
end.				

Inițializări implicite

Borland Pascal 7.0 face o inițializare automată a variabilelor de tipuri simple:

- tipurile numerice $\leftarrow 0$;
- tipul char $\leftarrow \#0$ (caracterul cu codul ASCII 0);
- tipul boolean $\leftarrow \text{FALSE}$.

Declararea constantelor

Constanta, spre deosebire de variabilă, nu-și poate modifica valoarea pe parcursul execuției programului. Constanta se declară în secțiunea **const** a programului și se memorează, de asemenea, în segmentul de date.

const

max = 100;

Constantele predefinite nu trebuie declarate în secțiunea **const**.

Exemple: Constante predefinite:

- Maxint = 32767;
- Maxlongint = 2147483647.

Expresii

Expresia este o succesiune de operanzi (variabile, constante, apeluri de funcții) legați prin operatori (aritmetici, relaționali, logici).

Evaluarea unei expresii

Orice expresie se evaluează de la stânga la dreapta, respectând prioritatea operatorilor.

prioritate
maximă

paranteze ()
not, +, - unari
and, *, /, div, mod
or, xor, +, -
=, <>, <, >, <=, >=

Tipul unei expresii

Tipul unei expresii este dat de tipul rezultatului expresiei. Trebuie să existe compatibilitate între tipul rezultatului expresiei și tipul variabilei căreia i se atribuie rezultatul expresiei.

program expresii;

var

a,b:integer;

c:longint;

begin

a := sqrt(16);

b := 10000*10000;

end.

Eroare de compilare **Type mismatch**, deoarece funcția de extragere a radicalului furnizează un rezultat de tip real.

În acest exemplu, funcția transmite rezultatul real **4.0** care are altă reprezentare internă decât valoarea întreagă **4**

Eroare de compilare **Constant out of range**, deoarece $10000 \times 10000 = 100000000$, iar domeniul de valori pentru tipul integer este $-32768 \dots 32767$.

8. Instrucțiuni de citire/scriere a datelor utilizând dispozitivele standard ale calculatorului: tastatura și monitorul

Instrucțiuni de citire a datelor — READ și READLN

Sintaxa instrucțiunilor de citire

Read(variabila₁, variabila₂, ..., variabila_n);

Readln(variabila₁, variabila₂, ..., variabila_n);

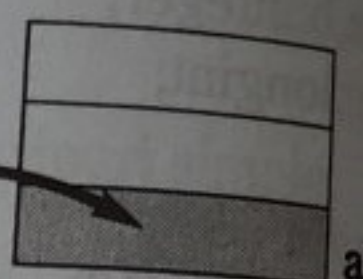
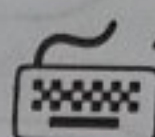
Datele de intrare, numerice sau caractere, se introduc de la tastatură. Pe măsură ce sunt tastate, datele se memorează într-o zonă specială de memorie, numită **buffer-ul tastaturii**. În buffer se vor memora codurile ASCII ale caracterelor tastate. Abia la apăsarea tastei <ENTER> informația memorată în acest buffer devine accesibilă programului. Instrucțiunea READ extrage codurile din buffer și le convertește intern conform tipului fiecărei variabile din lista de variabile descrisă în cadrul instrucțiunii de citire.

La apăsarea tastei <ENTER>, în buffer se vor memora două coduri:
#13 : *Car Return* — cursorul este adus la începutul liniei curente;
#10: *Line Feed* — cursorul este mutat pe linia următoare, în prima coloană.

Citirea unei singure valori de la tastatură

```
readln(var1) sau read(var1);
program citire_date1;
var
    a:byte;
begin
    readln(a);
end.
```

Valoarea tastată
este introdusă în
variabila **a**



Dacă s-a tastat 3 și <ENTER> buffer-ului tastaturii va arăta astfel:

3	#13	#10
---	-----	-----

.

Instrucțiunea **readln(a)** va introduce în variabila **a** valoarea tastată convertită la tipul **byte** al variabilei **a**.

În continuare, instrucțiunea **readln(a)** va extrage din buffer codurile introduse la apăsarea tastei <ENTER>, golind astfel buffer-ul.

Observații

1. Dacă în loc de **readln(a)** am fi utilizat instrucțiunea **read(a)**, valoarea 3 ar fi fost introdusă în variabila **a**, dar codurile <ENTER> ar fi rămas în buffer și ar fi fost extrase de următoarea instrucțiune **read**. Dacă nu luăm în considerare comportamentul instrucțiunii **read**, putem avea surpriza unor citiri incorecte ale datelor.

2. Dacă la lansarea în execuție a programului, tastăm următoarele: 3 <SPAȚIU> 6 <ENTER>, instrucțiunea **readln(a)** va extrage o singură valoare, 3, pe care o va atribui singurei variabile, **a**, după care va ignora valoarea 6 și va goli buffer-ul, extrăgând toate codurile până la <ENTER> inclusiv.

Instrucțiunea **read(a)** va extrage valoarea 3 pe care o va atribui variabilei **a**, după care va lăsa nemodificat restul buffer-ului tastaturii.

În exemplele următoare se tastează 3 <SPACE> 5 <ENTER> x

<pre>var c:char; a, b:integer; begin readln(a, b); read(c); end.</pre>	<pre>→ a ← 3 b ← 5 Golește buffer-ul. → c ← 'x'</pre>	<pre>var c:char; a, b:integer; begin read(a, b); read(c); end.</pre>	<pre>→ a ← 3 b ← 5 În buffer rămâne <ENTER>, astfel încât read(c) va extrage <ENTER> și c ← #13</pre>
--	---	--	--

3. Pentru a investiga starea buffer-ului tastaturii, dacă a fost sau nu apăsată o tastă, se poate utiliza funcția predefinită, **keypressed**. Funcția *keypressed* va transmite TRUE dacă a fost apăsată o tastă și FALSE dacă buffer-ul este vid. Ea este definită în unit-ul CRT (Turbo Pascal) sau în unit-ul WinCRT pentru varianta Windows.

4. Caracterul tastat nu apare în ecou pe ecran dacă utilizăm funcția **readkey**.

```
program citire_caracter;
uses CRT; {uses WinCRT;}
var
  c:char;
begin
  c := readkey;
end.
```

Citirea mai multor valori de la tastatură cu o singură instrucțiune de citire

Readln(variabila₁, variabila₂, ..., variabila_n);

Read(variabila₁, variabila₂, ..., variabila_n);

- Valorile se vor tasta respectând ordinea specificată în scrierea instrucțiunii.
- Valorile corespunzătoare diferitelor variabile se vor separa prin „caractere albe”: <SPAȚIU>, <TAB> sau <ENTER>.

Exemple:

Se examinează conținutul variabilelor după execuția fiecărei instrucțiuni de citire, utilizând meniul *Debug* → *Watch*. La fiecare apăsare a tastei <INSERT> se introduce în fereastră câte o variabilă. Programul se execută pas cu pas, apăsând tasta <F7>.

<pre>program date_1; var a:integer;b,c:real; begin readln(a, b, c); end.</pre>	Se tastează: 12<SPAȚIU>13.45<SPAȚIU> <SPAȚIU>9.87543E2 <ENTER>	Valorile variabilelor: $a \leftarrow 12$ $b \leftarrow 13.45$ $c \leftarrow 987.543$
--	--	---

<pre>program date_2; var a:integer; c:char; begin readln(c, a); end.</pre>	Se tastează: A123<ENTER>	Valorile variabilelor: $c = 'A'$ $a = 123$
	A<SPACE>123<ENTER>	$c = 'A'$ $a = 123$
	A<TAB>123<ENTER>	$c = 'A'$ $a = 123$
	AB123<ENTER>	Eroare: "Invalid numeric format" $c \leftarrow 'A'$, dar B123 nu este o valoare numerică.

<pre> program date_3; var a:integer; c:char; begin readln(a, c); end. </pre>	Se tastează:	Valorile variabilelor:
	67d<ENTER>	Eroare: "Invalid numeric format"
	67<SPACE>d<ENTER>	a = 67 c = ' '
<pre> program date_4; var a:byte; c1,c2,c3:char; begin read(c1,a, c2, c3); end. </pre>	67<TAB>d<ENTER>	a=67 c=#9 (codul tastei <TAB>)
	Se tastează:	Valorile variabilelor:
	-55<ENTER>	c1 = '-' a = 55 c2 = #13 c3 = #10

Instrucțiuni de afișare a datelor — WRITE și WRITELN

Instrucțiunea WRITE afișează:

1. Șiruri de caractere

- fără format

write('TESTE');

T E S T E

- cu format

write('TESTE':10);

Șirul este aliniat la dreapta în cadrul zonei de afișare de lățime egală cu 10 caractere.

T E S T E

2. Numere întregi

- fără format

write(123);

1 2 3

- cu format

write(123:10);

1 2 3

3. Numere reale

- cu format implicit

write(3271.2345);

Se folosește notația științifică cu 11 cifre semnificative.

3 . 2 7 1 2 3 4 5 0 0 0 E + 0 3

- cu format virgulă mobilă
Rw

write(-3271.2345:10);

$\pm x.xx...xE\pm yy$ utilizând w poziții pentru afișarea numărului

- 3 . 2 7 1 E + 0 3

- cu format
virgulă fixă
R:w:z

`write(3271.2365:10:2);`

$w = 10$ este lățimea zonei de afișare;

$z = 2$ este numărul de zecimale

3 2 7 1 . 2 4

				3	2	7	1	.	2	4									
--	--	--	--	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--

Observații

1. Dacă formatul de afișare este prea mic, el va fi extins automat în așa fel încât să se poată afișa întreaga valoare.
2. Dacă numărul de zecimale specificat la formatul în virgulă fixă este mai mic decât numărul de zecimale al valorii de afișat, ultima cifră zecimală afișată va fi rotunjită numai dacă prima cifră zecimală ignorată este mai mare sau egală cu 5.
3. Instrucțiunea **write** lasă cursorul după ultimul caracter afișat. **Writeln** afișează valorile specificate și apoi mută cursorul la începutul liniei următoare.

Situații de depășire

buffer
overflow

Fie următorul program care mărește cu 1 o valoare naturală tastată.

```
program depasire;
var a,b:byte;
begin
  write('Tastati un numar <= 255:');
  readln(a);
  b:=a+1;
  writeln('b= ',b);
  readln;
end.
```

Dacă tastăm valoarea 255 pentru variabila *a*, programul va afișa rezultatul **b = 0**, rezultat eronat, deoarece $255 + 1 = 256$.

Rezultatul eronat se datorează depășirii domeniului de valori al tipului *byte*: 0 .. 255.

255 se reprezintă intern prin 11111111.

Dacă adunăm 1, obținem:

$$\begin{array}{r} 11111111 + \quad \Leftrightarrow a + \\ \underline{ 1} \quad \quad \quad \frac{1}{b} \\ 100000000 \\ \quad \quad \quad \underbrace{}_b \end{array}$$

O variabilă de tip *byte* se reprezintă pe 1 byte, adică 8 biți. În acest caz, al nouălea bit, având valoarea 1, se va pierde și *b* va fi $00000000_{(2)} = 0_{(10)}$.

9. Instrucțiuni de citire/scriere a datelor utilizând fișiere text

Datele pot fi citite de pe disc (în loc de tastatură) și pot fi scrise pe disc (în loc de monitor).

Organizarea logică a unui fișier text

Fișierul text este format din caractere (codurile ASCII) grupate în linii de lungime variabilă, terminate cu marcajul <EOL>, codurile #13#10.

t	e	x	t	<EOL>			
e	x	e	m	p	l	e	<EOL>

Fișierul se încheie cu marcajul <EOF> (CTRL+Z), care va fi testat de funcția eof.

- Prelucrarea caracterelor se face SECVENȚIAL, adică în ordinea apariției caracterelor în fișier.
- Caracterul care urmează a fi prelucrat este indicat de către POINTERUL de FIȘIER. La deschiderea fișierului, pointerul de fișier este poziționat pe primul caracter.

Operații de bază asupra fișierelor text

Identificarea unui fișier text

- **Fișier extern** — prin numele fișierului pe disc, incluzând calea fișierului dacă acesta nu este în directorul curent
- **Fișier intern** — prin variabila fișier declarată în program (var f : text;). Asocierea dintre identificatorul extern și cel intern se face cu ajutorul procedurii assign:
assign(f , numele_fisierului);

Deschiderea unui fișier text

- **Fișier de intrare** (un fișier existent pe disc)
reset(f);
append(f); (* extinderea fișierului existent, prin adăugarea de caractere la sfârșitul fișierului *)
- **Fișier de ieșire** (se creează un fișier nou pe disc)
rewrite(f);

Operații care se pot efectua asupra caracterelor dintr-un fișier text

- **Operații de citire secvențială**
read(f, var₁, var₂, ..., var_n); readln(f, var₁, var₂, ..., var_n);

Mecanismul de funcționare a instrucțiunilor read și readln este identic pentru tastatură și fișiere text.

- Operații de scriere secvențială

`write(f, var1, var2, ..., varn); writeln(f, var1, var2, ..., varn);`

- Ștergerea unui fișier text

`erase(f);`

- Redenumirea unui fișier text

`rename(f, noul_nume);`

- Închiderea unui fișier text

`close(f);`

Funcții predefinite

- **eol(f)** întoarce valoarea logică TRUE dacă următorul caracter din fișier este marcajul de sfârșit de linie;

- **eof(f)** întoarce valoarea logică TRUE dacă următorul caracter din fișier este marcajul de sfârșit de fișier;

- **seekeof(f)** întoarce valoarea logică TRUE dacă următorul caracter din fișier este marcajul de sfârșit de fișier, avansând însă, peste marcasele de sfârșit de linie, spații și <TAB>.

Exemplu: Se citesc din fișierul text '**date.in**' două numere naturale. Se scrie suma numerelor în fișierul text '**date.out**'.

```
program fisiere_text;
var
  a, b, s:byte;
  f, g:text;
begin
  assign(f, 'date.in');
  reset(f);
  assign(g, 'date.out');
  rewrite(g);
  readln(f, a, b);
  s := a+b;
  writeln(g, 'Suma= ', s);
  close(f);
  close(g);
end.
```

Fișierul text '**date.in**' se poate crea cu editorul din mediul integrat de programare Pascal, ca orice program sursă.

Dacă fișierul de intrare are următoarea structură:

2 5 <EOL>

atunci $a \leftarrow 2$ $b \leftarrow 5$

și fișierul de ieșire va conține o singură linie:

Suma = 7 <EOL>

Observații

1. Dacă se execută procedura `rewrite(f)` asupra unui fișier existent pe disc, conținutul inițial al fișierului se va șterge.
2. Dacă uităm să închidem fișierul de ieșire putem pierde datele de ieșire.

Stil de programare

Arta programării implică și arta organizării datelor. Să nu uităm că datele reprezintă unul dintre ingredientele din care se „prepară” un bun program.

Tipurile datelor prelucrate de program trebuie alese cu multă atenție pentru a nu ajunge la situații de depășire.

Paza bună trece primejdia rea!

Verificați corectitudinea introducerii datelor în memoria internă înainte de a trece la implementarea algoritmului.

10. Bifurcații și intersecții în traficul unui program

Pe măsură ce programele devin mai complexe apar situații în care programul trebuie „să ia o decizie”: să efectueze un anumit set de instrucțiuni, dacă o condiție este adevărată, și alt set de instrucțiuni, dacă aceasta este falsă.

Pentru implementarea algoritmilor decizionali se utilizează instrucțiunile IF și CASE.

Instrucțiunea IF — o bifurcație în traficul programului

Pentru implementarea algoritmilor decizionali alternativi se folosește instrucțiunea IF.

Sintaxa instrucțiunii IF

Instrucțiunea IF are două formate:

Formatul 1:

```
if <expresie_logică> then  
    <instrucțiunea_1>  
else  
    <instrucțiunea_2>;
```

Formatul 2:

```
if <expresie_logică> then  
    <instrucțiunea_1>;
```

Mecanismul de funcționare a instrucțiunii IF

1. Se evaluează expresia logică. Dacă valoarea expresiei logice este **TRUE**, se execută **instrucțiunea_1**, iar dacă este **FALSE**, se execută **instrucțiunea_2**.
2. Dacă **instrucțiunea_2** este vidă, se folosește **formatul 2** al instrucțiunii IF.
3. Înainte de **else** nu se pune caracterul „;”.
4. **Instrucțiunea_1** sau **instrucțiunea_2** pot fi la rândul lor instrucțiuni IF, obținând astfel instrucțiuni **IF imbricate**, adevărate cascade de condiții.
5. **Instrucțiunea_1** sau **instrucțiunea_2** pot fi **instrucțiuni compuse**, delimitate prin cuvintele rezervate *begin* și *end*.

Formatul 1:

```
if <expresie_logică> then
    begin
        <instrucțiunea_1>
    end
else
    begin
        <instrucțiunea_2>
    end;
```

Formatul 2:

```
if <expresie_logică> then
    begin
        <instrucțiunea_1>
    end;
```

6. **Else** se asociază întotdeauna cu ultimul **if** care nu a fost încă asociat. Această asociere poate fi modificată utilizând delimitatorii *begin* și *end*.

Exemplul 1:

```
if <expresie_logică_1> then
    if <expresie_logică_2> then
        <instrucțiunea_2>
    else
        <instrucțiunea_3>;
```

Exemplul 2:

```
if <expresie_logică_1> then
    BEGIN
        if <expresie_logică_2> then
            <instrucțiunea_2>
        END
    else
        <instrucțiunea_3>;
```

Problema 1. Număr par sau impar?

Se citește de la tastatură un număr natural n . Să se determine dacă numărul tastat este par sau impar, afișând un mesaj corespunzător. **Restricții:** $0 \leq n \leq 2\,000\,000\,000$.

1. Analiza problemei

1.1. Datele problemei

a) Date de intrare: cele din tabelul 33.

Tabelul 33

Nr. test	Nume	Ce reprezintă?	Domeniul de valori	Tipul variabilei
1.	n	numărul tastat	$0 \leq n \leq 2\,000\,000\,000$	longint

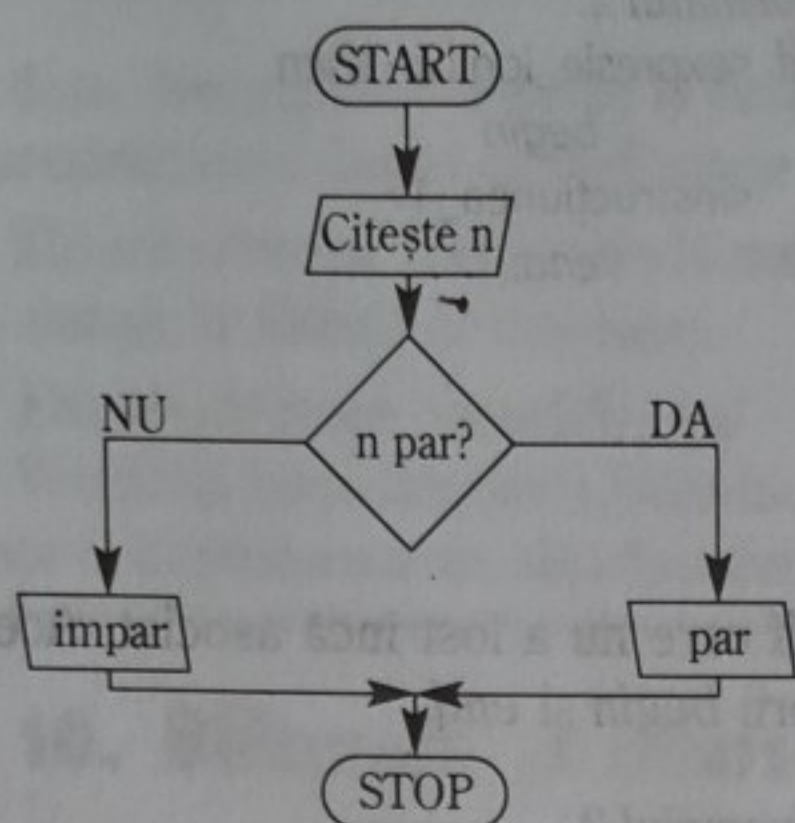
b) Date de ieșire: mesajul „PAR” sau „IMPAR”.

1.2. Funcția programului. „Ce” trebuie să facă programul și „cum”?

„Ce?” Programul trebuie să verifice dacă numărul natural tastat este par sau impar.

„Cum?” Un număr natural este par dacă restul împărțirii numărului la 2 este 0 și este impar dacă restul împărțirii numărului la 2 este 1 (diferit de 0).

2. Descrierea algoritmului



Observații

1. Testarea parității unui număr natural se poate face și cu ajutorul funcției `odd`, care returnează `TRUE` dacă numărul este impar și `FALSE` dacă numărul este par.

Instrucțiunea `if` din programul de mai sus poate fi înlocuită cu următoarea secvență:

```
if odd(n) then
    write('IMPAR')
else
    write('PAR');
```

2. Instrucțiunea finală `readln` așteaptă apăsarea tastei <Enter> pentru a se realiza comutarea din *User Screen* (ecranul cu rezultate) în ecranul de editare a programului. Dacă nu scriem această instrucțiune la finalul programului va trebui să apăsăm simultan tastele <Alt> și <F5> pentru a putea „viziona” rezultatele execuției programului.

4. Testarea programului

Pentru a verifica dacă programul funcționează corect, va trebui să pregătim un set de date de test.

Tabelul 34

Nr. test	n	Rezultat
1.	4	PAR
2.	7	IMPAR
3.	99999	IMPAR

Observație. Dacă am fi ales pentru variabila `n` tipul `integer`, programul nu ar fi trecut testul 3, deoarece cea mai mare valoare de tip `integer` este 32767.

3. Implementarea algoritmului în limbajul de programare Pascal

```
program problema1;
var
    n:longint;
begin
    write('n= '); readln(n);
    if n mod 2 = 0 then
        write('PAR')
    else
        write('IMPAR');
    readln;
end.
```


Problema 2. Cel mai înalt elev

Se citesc de la tastatură înălțimile elevilor Dragoș și Andrei. Să se afișeze numele celui mai înalt elev. **Restricții:** Înălțimile elevilor se exprimă în centimetri și nu pot depăși 200 de cm.

1. Analiza problemei

1.1. Datele problemei

a) Date de intrare:
cele din tabelul 35.

Numele unei variabile trebuie ales astfel încât să sugereze ceea ce reprezintă. Am ales **d** de la Dragoș, dar la fel de bine puteam numi variabila **Dragos** sau **dragos**.

Tabelul 35

Nr. test	Numele	Ce reprezintă?	Domeniul de valori	Tipul variabilei
1.	d	înălțimea lui Dragoș	$0 \leq d \leq 200$	byte
2.	a	înălțimea lui Andrei	$0 \leq a \leq 200$	byte

b) Date de ieșire: numele celui mai înalt elev.

1.2. Funcția programului

Programul trebuie să compare înălțimile celor doi elevi și să afișeze numele celui mai înalt elev.

2. Descrierea algoritmului

```
început problema 2
  citește d
  citește a
  dacă d > a atunci
    scrie 'Dragos'
  altfel
    scrie 'Andrei'
sf dacă
sfârșit problema 2
```

3. Implementarea algoritmului în limbajul de programare Pascal

```
program problema2;
var d, a:byte;
begin
  write('Inaltimea lui
Dragos:');readln(d);
  write('Inaltimea lui
Andrei:');readln(a);
  if d > a then
    write('Dragos')
  else
    write('Andrei');
  readln;
end.
```

4. Testarea programului

Vom alege datele de test, astfel încât să acopere toate situațiile posibile:

Tabelul 36

Nr. test	d	a	Rezultat
1.	150	145	Dragos
2.	145	150	Andrei
3.	150	150	Dragos si Andrei

Executând programul pentru fiecare test, se constată că la testul 3 programul va afișa doar 'Andrei'.

Înlocuind variabilele **d** și **a** cu valorile tastate, calculatorul va evalua în cadrul instrucțiunii **if** expresia logică: $150 > 150$. Deoarece expresia este falsă, se va executa instrucțiunea de pe ramura **else**, respectiv, **write('Andrei')**.

Observație. Instrucțiunea de pe ramura **else** se execută când $d \leq a$. Pentru a obține răspunsul corect la testul 3 trebuie să modificăm programul astfel:

```
program problema2;
var   d, a:byte;
begin
    write('Inaltimea lui Dragos:'); readln(d);
    write('Inaltimea lui Andrei:'); readln(a);
    if d > a then
        write('Dragos')
    else
        if d < a then
            write('Andrei')
        else
            write('Dragos si Andrei');
    readln;
end.
```

Deoarece instrucțiunea de pe ramura **else** este tot o instrucțiune **if**, am creat un „mic arbore”, utilizând instrucțiuni **if imbricate**.

Să reluăm testul 3 pentru a doua variantă a programului „Cel mai înalt elev”. Răspunsul este acum 'Dragoș și Andrei'.

Problema 3. Maxim

Se citesc de la tastatură trei numere naturale, formate din cel mult 9 cifre. Să se afișeze valoarea maximă a celor trei valori tastate.

Exemplu: $a = 13, b = 5, c = 23$

Se va afișa: $\max(13, 5, 23) = 23$.

1. Analiza problemei

1.1. Datele problemei

a) Date de intrare: cele din tabelul 37.

Tabelul 37

Nr. test	Numele	Ce reprezintă?	Domeniul de valori	Tipul variabilei
1.	a	primul număr tastat	$0 \leq a \leq 999999999$	longint
2.	b	al doilea număr tastat	$0 \leq b \leq 999999999$	longint
2.	c	al treilea număr tastat	$0 \leq c \leq 999999999$	longint

b) Date de ieșire: **max** = cea mai mare valoare dintre valorile variabilelor a, b, c.

1.2. Funcția programului

Programul trebuie să compare cele trei valori și să o determine pe cea mai mare, adică valoarea maximă.

2. Descrierea algoritmului

început problema 3

 citește a, b, c

 max ← a

 dacă b > max atunci

 max ← b

 sf dacă

 dacă c > max atunci

 max ← c

 sf dacă

 scrie max

sfârșit problema 3

3. Implementarea algoritmului în limbajul de programare Pascal

program problema3;

var a, b, c, max: longint;

begin

write('a, b, c= ');

readln(a, b, c);

 max := a;

if b > max **then**

 max := b;

if c > max **then**

 max := c;

write('max(' , a, ' , ' , b, ' , ' , c, ') = ' , max);

readln;

end.

4. Testarea programului

Tabelul 38

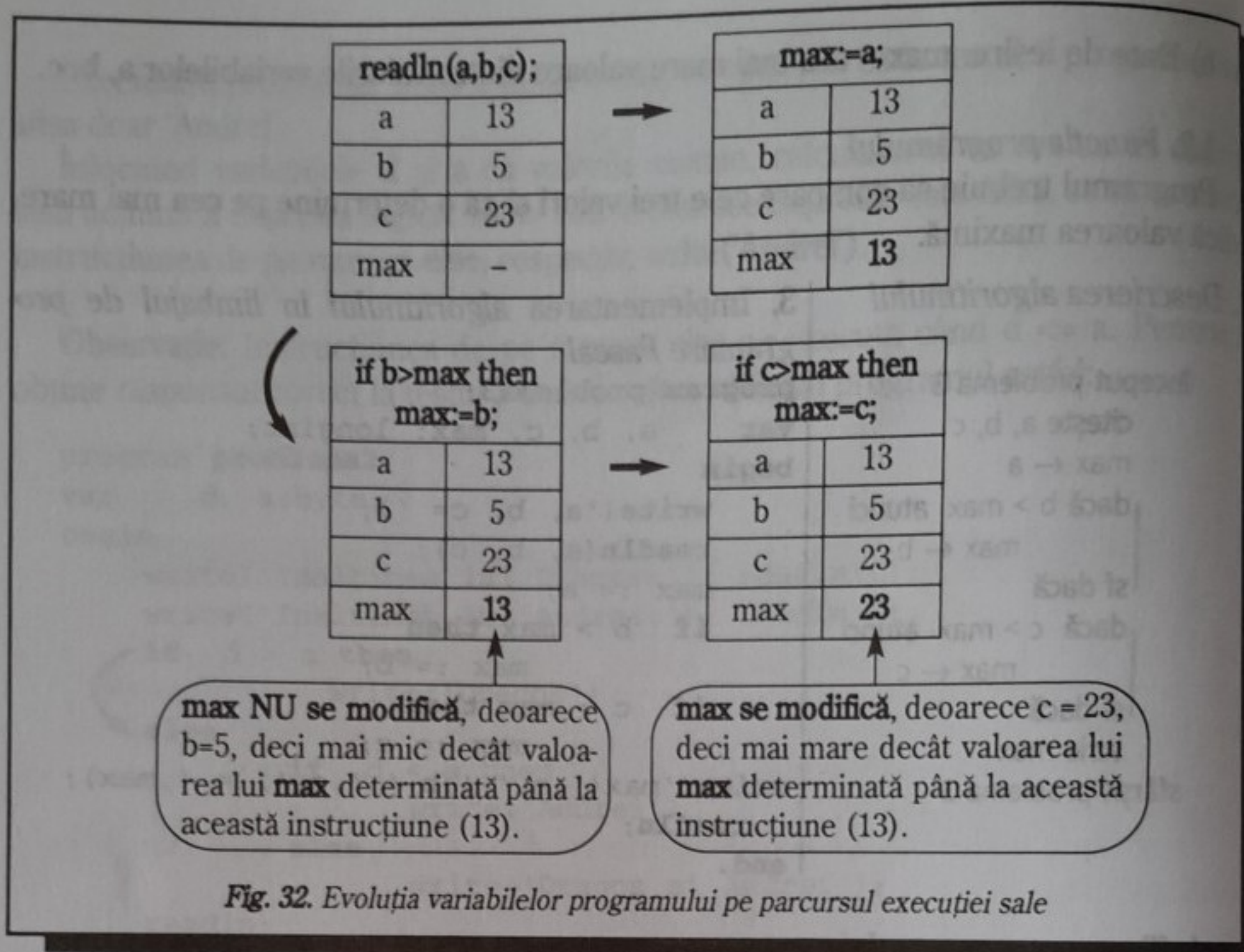
Nr. test	a	b	c	max	Observații
1.	13	5	23	23	toate valorile sunt distincte
2.	53	55	54	55	
3.	100000	3	99999	100000	
4.	999	999	900	999	câte două valori egale
5.	11	13	13	13	
6.	111	555	111	555	
7.	9	9	9	9	toate valorile egale

Să urmărim cum se schimbă conținutul variabilei **max** în timpul execuției testului 1.

Tehnici de depanare. Execuția pas cu pas a unui program

Vom adăuga în fereastra Watch (Meniul Debug) variabilele pe care dorim să le examinăm pe parcursul execuției programului. Adăugarea unei noi variabile în fereastra Watch se poate face apăsând tasta <Insert>.

Executăm pas cu pas programul prin acționarea tastei <F7>, urmărind în fereastra Watch cum se schimbă conținutul variabilelor **a**, **b**, **c**, **max** la execuția fiecărei instrucțiuni. Evoluția variabilelor programului la execuția pas cu pas a acestuia se poate urmări în figura 32.



Observație. Algoritmul descris anterior este valabil pentru oricâte numere dorim și, de asemenea, pentru determinarea minimului (se schimbă doar sensul inegalității, adică $<$ în loc de $>$).

Problema 4. Cadoul

Irina vrea să cumpere două cărți pentru ziua de naștere a prietenei ei, Ana. Dar Irina trebuie să aleagă două cărți dintre cele trei cărți pe care și le dorește Ana.

Știind că Irina dispune de s lei și că dorește să-i rămână cât mai mulți lei după cumpărarea celor două cărți, astfel încât să poată cumpăra și niște flori frumoase, scrieți un program prin care să o ajutați să aleagă cele două cărți.

Suma de care dispune ea, precum și prețurile celor trei cărți se vor introduce de la tastatură. Programul va afișa numerele de ordine ale cărților din care se compune cadoul. **Restricții:** toate valorile sunt numere naturale mai mici sau egale cu 1000000.

1. Analiza problemei

1.1. Datele problemei

a) Date de intrare: cele din tabelul 39.

Tabelul 39

Nr. test	Numele variabilei	Ce reprezintă?	Domeniul de valori	Tipul variabilei
1.	s	suma de lei de care dispune Irina	$0 \leq s \leq 1000000$	longint
2.	p1	prețul primei cărți	$0 \leq p1 \leq 1000000$	longint
3.	p2	prețul celei de-a doua cărți	$0 \leq p2 \leq 1000000$	longint
4.	p3	prețul celei de-a treia cărți	$0 \leq p3 \leq 1000000$	longint

b) Date de ieșire: două numere, reprezentând numerele de ordine ale cărților alese.

1.2. Funcția programului

Ca într-un joc Lego putem împărți problema în subprobleme, pe care le vom îmbina apoi pentru rezolvarea problemei.

- *Subproblema 1.* Vom calcula prețurile tuturor pachetelor cu două cărți care se pot forma utilizând cele trei cărți.
- *Subproblema 2.* Determinăm pachetul cel mai ieftin, deoarece, alegând cel mai ieftin pachet, Irinei îi va rămâne cea mai mare sumă pentru flori.

2. Descrierea algoritmului

P1. Se citesc datele de intrare: s, p1, p2, p3.

P2. Se calculează prețul fiecărui pachet cu două cărți:

pachet1 = p1 + p2;

pachet2 = p1 + p3;

pachet3 = p2 + p3;

P3. Se determină valoarea minimă dintre **pachet1**, **pachet2** și **pachet3**.

P4. Dacă valoarea minimă determinată la pasul **P3** este mai mică (sau egală) decât suma **s** de care dispune Irina, se vor afișa numerele de ordine ale cărților din pachet.

3. Implementarea algoritmului în limbajul de programare Pascal

```

program problema4;
var s, p1, p2, p3, pachet1, pachet2, pachet3, min:longint;
begin
    write('Suma Irinei:'); readln(s);
    write('Preturile celor 3 carti:');
    readln(p1, p2, p3);

    pachet1 := p1 + p2;
    pachet2 := p1 + p3;
    pachet3 := p2 + p3;

```



```

min:=pachet1;
if pachet2 < min then min := pachet2;
if pachet3 < min then min := pachet3;

if s>=min then
  if min = pachet1 then writeln('1 2')
  else
    if min = pachet2 then writeln('1 3')
    else writeln('2 3');
  readln; end.

```

4. Testarea programului

Tabelul 40

Nr. test	S	p1	p2	p3	pachet 1	pachet 2	pachet 3	min	Rezultat
1.	100	30	20	40	50	70	60	50	1 2
2	200	150	100	120	250	270	220	220	Insuficient
3.	100	30	20	30	50	60	50	50	1 2 sau 2 3
4.	100	30	30	30	60	60	60	60	1 2 sau 1 3 sau 2 3
5.	100	70	60	40	130	110	100	100	2 3

Observație. Alegând datele de test astfel încât să acopere toate situațiile posibile, observăm că apar situații care nu au fost tratate în algoritmul de mai sus:

1. Situația în care cel mai ieftin pachet este mai scump decât suma de care dispune Irina. În acest caz trebuie afișat un mesaj, de genul, 'Insuficient' (*testul nr. 2*).

Un bun programator trebuie să aibă pregătit un mesaj pentru fiecare situație care poate apărea la execuția programului.

2. *Testele 3 și 4* scot în evidență faptul că pot exista mai multe soluții. De exemplu, în *testul 3* soluțiile posibile sunt: pachetul format din cărțile 1 și 2 sau pachetul format din cărțile 2 și 3. În astfel de situații, dacă în enunțul problemei nu s-a făcut nici o precizare de tipul „dacă există mai multe soluții, se afișează una dintre ele”, se vor afișa toate soluțiile posibile.

Ținând cont de aceste observații va trebui să modificăm programul de mai sus. Forma lui finală, de 100 de puncte, va fi următoarea:

```

program problema4;
var
  s, p1, p2, p3, pachet1, pachet2, pachet3, min:longint;
begin

```



```

write('Suma Irinei: '); readln(s);
write('Preturile celor 3 carti: ');
readln(p1, p2, p3);

pachet1 := p1 + p2;
pachet2 := p1 + p3;
pachet3 := p2 + p3;

min:=pachet1;
if pachet2 < min then min := pachet2;
if pachet3 < min then min := pachet3;

if s >= min then
begin
    if min = pachet1 then writeln('1 2');
    if min = pachet2 then writeln('1 3');
    if min = pachet3 then writeln('2 3');
end
else write('Insuficient');
readln;
end.

```

Pentru **claritatea** programului se poate lăsa câte o linie albă între subprobleme.

Probleme propuse

Completați programul **Cadou** astfel ca atunci când prețul celui mai ieftin pachet coincide cu suma de care dispune Irina să apară mesajul „Irina nu poate cumpăra flori!” (testul 5).

Problema 5. Cele mai mari două numere

Să se determine cele mai mari două numere dintre trei numere naturale introduse de la tastatură.

1. Analiza problemei

1.1. Datele problemei

- a) Date de intrare: **a, b, c** — numerele tastate
- b) Date de ieșire: **max1, max2** — cele mai mari numere

1.2. Funcția programului

Se vor determina cele mai mari două numere dintre trei numere tastate, utilizând două variabile: max1 și max2.

2. Descrierea algoritmului

P1. Se citesc datele de intrare **a**, **b**, **c**.

P2. Se initializează **max1 = a** și **max2 = b**.

P3. Se verifică poziția variabilei **b** față de **max1** și **max2**.

- Dacă **b** este mai mare decât maximul presupus inițial (**max1**), **max1** va prelua valoarea lui **b**, dar nu înainte de a se salva în **max2** vechea valoare a lui **max1**.
- Dacă **b** nu este mai mare decât **max1**, **max1** va rămâne neschimbat, dar e posibil ca **b** să fie mai mare decât **max2**. Dacă **b** este mai mare decât **max2**, **max2** va prelua valoarea lui **b**, dacă nu, **max2**, la fel ca și **max1**, vor rămâne neschimbate.

P4. Operațiile de la pasul P3 se repetă pentru variabila **c**.

P5. Se afișează valorile variabilelor **max1** și **max2**.

3. Implementarea algoritmului în limbajul de programare Pascal

```
program problema5;
var a, b, c, max1, max2: longint;
begin
    write('a,b,c='); readln(a,b,c);
    max1 := a; max2 := b;
    if b > max1 then
        begin
            max2 := max1; max1 := b;
        end
    else if b > max2 then max2 := b;

    if c > max1 then
        begin
            max2 := max1; max1 := c;
        end
    else if c > max2 then max2 := c;
    writeln(max1, ' ', max2);
    readln;
end.
```

4. Testarea programului

Tabelul 41

Nr. test	a	b	c	max1	max2
1.	4	5	8	8	5
2.	9	7	8	9	8
3.	9	9	8	9	9
4.	7	8	8	8	8
5.	9	9	9	9	9

Problema 6. Forme de relief

Se introduc de la tastatură trei numere naturale, mai mici decât 10000. Să se rearanjeze cele trei numere pe ecran astfel încât să formeze următoarele forme de relief:

PODIȘ



MUNTE



1. Analiza problemei

1.1. Datele problemei

- a) Date de intrare: **a, b, c** — numerele tastate
- b) Date de ieșire: **max1, max2** — cele mai mari numere; $\max1 \leq \max2$
min — cel mai mic număr

1.2. Funcția programului

Se vor determina cele mai mari două numere și cel mai mic dintre cele trei numere tastate.

2. Descrierea algoritmului

P1. Se citesc datele de intrare **a, b, c**.

P2. Se determină cele mai mari două numere și cel mai mic dintre cele trei numere tastate

P3. Se verifică dacă **max1, max2** și **min** pot fi afișate într-o anumită ordine astfel încât să formeze un „PODIȘ”. Dacă $\max1 = \max2 = \min$ atunci se afișează mesajul „PODIȘ”, dacă nu, se afișează mesajul „Imposibil”.

P4. Se verifică dacă **max1, max2** și **min** pot fi afișate într-o anumită ordine încât să formeze un „MUNTE”. Dacă $\max1 < \max2$, atunci există un vârf, vârful muntelui. Dacă $\max2 < \min$, atunci există două forme posibile: $\max2, \max1, \min$ sau $\min, \max1, \max2$. Dacă nu este posibil să construim această formă de relief se afișează mesajul „Imposibil”.

3. Implementarea algoritmului în limbajul de programare Pascal

```
program problema6;
var a,b,c,max1,max2,min:longint;
begin
    write('a, b, c= ');readln(a, b, c);
    max1 := a; max2 := b; min := a;
    if b > max1 then
        begin
            max2 := max1; max1:= b;
        end
    else if b > max2 then max2 := b;
    if b < min then min := b;
    if c > max1 then
        begin
            max2 := max1; max1:= c;
        end
    end
```



```

else if c > max2 then max2 := c;
if c < min then min := c;
write('PODIS: ');

```

AND se utilizează în cazul condițiilor simultan adevărate.

```

if (max1 = max2) and (max2 = min) then
    write(max1, ' ', max2, ' ', min)
else write('Imposibil');
writeln;
write('MUNTE:');
if (max1 <> max2) then
    begin
        write(min, ' ', max1, ' ', max2);
        if (max2 <> min) then
            write(' sau ', max2, ' ', max1, ' ', min);
        end
    else write('Imposibil');
    readln;
end.

```

4. Testarea programului

Tabelul 42

Nr. test	a	b	c	Rezultat
1.	4	5	8	PODIS: Imposibil MUNTE: 4 8 5 sau 5 8 4
2.	9	9	9	PODIS: 9 9 9 MUNTE: Imposibil
3.	8	6	8	PODIS: Imposibil MUNTE: Imposibil
4.	8	5	5	PODIS: Imposibil MUNTE: 5 8 5

Probleme propuse

Extindeți programul de mai sus pentru următoarele forme de relief:

vale



povârniș



râpă



Problema 7. Pinochio (100 de puncte)

În fiecare zi lucrătoare din săptămână, Pinochio spune câte o minciună datorită căreia îi crește nasul cu câte p cm pe zi. Sâmbăta și duminica, când vine bunicul Gepeto acasă, pentru a nu-l supăra prea tare, Pinochio reușește să nu spună nici o minciună, ba chiar uitându-se în oglindă

observă că în fiecare din aceste zile lungimea nasului său scade cu câte 1 centimetru pe zi. Când începe o nouă săptămână, rămânând singur acasă, Pinochio continuă șirul minciunilor.

Cerință: Care este dimensiunea nasului lui Pinochio după k zile, știind că inițial nasul său măsura n centimetri?

Date de intrare: De la tastatură se citesc n , p și k .

Date de ieșire: Pe ecran se va afișa un singur număr natural, adică numărul de centimetri cerut de problemă.

Restricții: $1 \leq n \leq 1000$, $1 \leq k \leq 256$, $1 \leq p \leq 100$

Exemplu:

Pentru $n = 2$ $p = 1$ $k = 8$ se va afișa 6 cm

Observație

Pentru orice exemplu zilele încep cu luni. Pe exemplul anterior zilele sunt: luni, marți, miercuri, joi, vineri, sâmbătă, duminică, luni.

Timp maxim de execuție: 1 secundă/test.

(ONIGIM¹ — Olimpiada Națională de Informatică pentru Gimnaziu
Faza interjudețeană, martie 2003, clasa a V-a)

1. Analiza problemei

1.1. Datele problemei

a) Date de intrare: cele din tabelul 43

Tabelul 43

Nr. test	Nume	Ce reprezintă?	Domeniul de valori	Tipul variabilei
1.	n	lungimea inițială a nasului lui Pinochio	$1 \leq n \leq 1000$	integer
2.	p	numărul de cm cu care crește nasul lui Pinochio în zilele lucrătoare	$1 \leq p \leq 100$	integer
3.	k	numărul de zile după care se calculează lungimea nasului lui Pinochio	$1 \leq k \leq 256$	integer

b) Date de ieșire: **lungime_nas**=lungimea nasului lui Pinochio după k zile

1.2. Funcția programului

Inițial nasul lui Pinochio are n cm. Calculul lungimii nasului lui Pinochio după k zile începe cu ziua de luni din prima săptămână.

Pentru a calcula lungimea nasului lui Pinochio după k zile va trebui să rezolvăm următoarele subprobleme:

1. Cu câți centimetri va crește nasul lui Pinochio într-o săptămână completă?

Știind că în zilele lucrătoare (luni, marți, miercuri, joi, vineri), primele 5 zile din săptămână, nasul îi crește cu p cm/zi, iar sâmbăta și duminica nasul se scurtează cu câte 1 cm/zi, nasul lui Pinochio va crește într-o săptămână completă cu $(p \cdot 5 - 2)$ cm.

¹ **ONIGIM** – Olimpiada Națională de Informatică pentru Gimnaziu, organizată de Ministerul Educației și Cercetării, începând cu anul 2001. Fazele naționale au avut loc la Gălăciuc și Focșani, dar selecția lotului național al fiecărui județ cuprinde următoarele faze: faza pe școală, faza pe sector (București), faza pe municipiu și faza pe Internet (interjudețeană).

2. Câte săptămâni complete (formate din 7 zile) sunt cuprinse în cele k zile?
 Numărul de săptămâni cuprinse în cele k zile este egal cu partea întreagă a câtului obținut la împărțirea lui k la 7, respectiv $(k \text{ div } 7)$.

3. Din câte zile este formată săptămâna incompletă?

Numărul de zile din săptămâna incompletă este egal cu restul obținut la împărțirea lui k la 7, respectiv $(k \text{ mod } 7)$. Acest rest poate fi și zero, dacă numărul de zile k este multiplu de 7.

Asamblând subproblemele de mai sus, asemenea unor blocuri dintr-un joc LEGO, obținem răspunsul cerut.

2. Descrierea algoritmului

început Pinochio

citește n, p, k

$\text{lungime\_nas} = n$ {Inițializarea lungimii nasului}

$L_{\text{sapt}} = p * 5 - 2$

$\text{nr\_săptămâni} = k \text{ div } 7$

$\text{rest\_zile} = k \text{ mod } 7$

$\text{lungime\_nas} = \text{lungime\_nas} + \text{nr\_săptămâni} * L_{\text{sapt}}$

dacă $\text{rest\_zile} \leq 5$

atunci

$\text{lungime\_nas} = \text{lungime\_nas} + \text{rest\_zile} * p$

altfel { $\text{rest\_zile} = 6$ }

$\text{lungime\_nas} = \text{lungime\_nas} + 5 * p - 1$

sf dacă

scrie lungime_nas

sf Pinochio

Probleme propuse

Încercați să transcrieți algoritmul de mai sus în limbajul Pascal și verificați corectitudinea programului cu următoarele teste:

Tabelul 44

Nr. test	n	p	k	Date de ieșire	Punctaj
1.	1	1	1	2	5 puncte
2.	1	10	6	50	10 puncte
3.	2	8	7	40	10 puncte
4.	4	6	75	314	10 puncte
5.	9	3	50	103	10 puncte
6.	2	4	146	381	10 puncte
7.	3	3	210	393	10 puncte
8.	7	5	226	753	10 puncte
9.	10	100	255	18238	10 puncte
10.	1000	100	256	19328	15 puncte
				TOTAL =	100 puncte

Instrucțiunea CASE — o intersecție în traficul programului

Pentru implementarea algoritmilor decizionali cu selecție se utilizează instrucțiunea CASE.

Sintaxa instrucțiunii CASE

Sintaxa instrucțiunii CASE este următoarea:

CASE <selector> OF

c_1 :<instrucțiunea_1>;

c_2 :<instrucțiunea_2>;

...

c_n :<instrucțiunea_n>;

else <instrucțiunea_else>;

END;

Mecanismul de funcționare a instrucțiunii CASE

1. Selectorul poate fi o variabilă sau o expresie de tip ordinal.
2. $c_1 \dots c_n$ sunt constante de același tip cu selectorul. În același CASE nu pot exista două constante cu valori identice.

3. Se compară, pe rând, valoarea selectorului cu valorile constantelor $c_1 \dots c_n$. Dacă valoarea selectorului coincide cu una dintre constante, se execută instrucțiunea asociată constantei, după care se încheie execuția instrucțiunii CASE și se continuă cu următoarea instrucțiune după CASE.

Dacă nu este întâlnită nici o coincidență, se execută <instrucțiunea_else> (dacă există alternativa **else**) și se continuă cu următoarea instrucțiune după CASE.

4. În cazul în care, pentru mai multe constante, se dorește executarea aceleiași prelucrări, atunci constantele respective se pot grupa într-o listă în care constantele vor fi separate prin **virgulă** sau prin '..', în cazul în care lista este formată din valori consecutive.

Exemplu:

case luna of

1,3,5,7,8,10,12: write('31 de zile');

4,6,9,11: write('30 de zile');

2: write('28 sau 29 de zile');

else write('Numărul lunii este incorect!');

end;

case nota of

8..10: write('Foarte bine');

6,7: write('Bine');

5: write('Suficient');

1..4: write('Insuficient');

else write('Notă incorectă!');

end;

IF versus CASE

1. CASE testează doar egalitatea, în timp ce IF poate să evalueze orice tip de expresie logică sau relațională.
2. Orice instrucțiune CASE poate fi scrisă cu ajutorul unor instrucțiuni IF imbricate.
3. Se pot scrie și instrucțiuni CASE imbricate.

Problema 8. Meniu cu ... operații aritmetice

Se știe că recordul privind viteza de efectuare a operațiilor aritmetice aparține calculatorului, care poate executa milioane de operații elementare pe secunde.

Scrieți un program care execută operații de adunare, scădere, înmulțire și împărțire, operandii fiind două numere reale, x și y .

Alegerea unei operații dintre cele patru operații posibile se face cu ajutorul unui meniu.

```
program problema8;
var optiune:byte; eroare:boolean;
    x,y,rezultat:real;
begin
    eroare:=false;

    write('Tastati doua numere:');readln(x,y);
    (* Afisarea meniului cu operatii aritmetice *)
    writeln('1. Adunare');
    writeln('2. Scadere');
    writeln('3. Inmultire');
    writeln('4. Impartire');
    write('Tastati optiunea (1..4):'); readln(optiune);
    case optiune of
    1: rezultat := x + y;
    2: rezultat := x - y;
    3: rezultat := x * y;
    4: if y <> 0 then rezultat := x / y
        else eroare := true;
    else eroare := true;
    end;
    if not eroare then
        writeln('Rezultat= ', rezultat:0:2)
    else
        writeln('Eroare date de intrare');
    readln;
end.
```

Variabila **eroare** acționează ca o „variabilă martor” la producerea unui eveniment, luând valoarea TRUE la producerea evenimentului.

Dacă nu am fi pus condiția $y \neq 0$ programul s-ar fi terminat forțat cu eroarea de execuție **Division by zero**, pentru cazul $y = 0$.

Problema 9. Orologiul

Un orologiu indică ora h și m minute. Ce oră va indica orologiul peste un minut? Restricții:
 $0 \leq h \leq 23$; $0 \leq m \leq 59$.

Vom începe prin determinarea tuturor testelor care să facă din acest program un program ireproșabil, adică un program de 100 de puncte.

Tabelul 45

Nr. test	Ora curentă		Ora indicată peste 1 minut	
	h	min	h	min
1.	9	60		Eroare
2.	24	3	Eroare	
3.	8	30	8	31
4.	0	0	0	1
5.	8	59	9	0
6.	23	59	0	0

În testele 1 și 2, minutele, respectiv orele, sunt niște valori incorecte, care nu țin seama de restricțiile specificate în enunțul problemei. În aceste situații programul trebuie să afișeze un mesaj de eroare și să abandoneze execuția.

Programul va începe cu operația de **validare a datelor de intrare**.

Testele 3 și 4 reprezintă cazul general, iar testele 5 și 6 reprezintă situațiile particulare, când se modifică ora inițială.

Varianta următoare de program „evită” instrucțiunile if utilizând operatorii aritmetici **mod** și **div**.

```

program orologiu;
var h,m,total_min:integer;
begin
    write('Tastati ora sub forma: h min:');
    readln(h,m);
    if (h > 23) or (m > 59) then      (* VALIDAREA DATELOR *)
        write('Eroare date de intrare')
    else
        BEGIN
            total_min := m + 1;
            m := total_min mod 60;
            h := h + total_min div 60;
            h := h mod 24;
            write('Orologiul bate ', h, 'h ', m, 'min');
        END;
    readln;
end.

```


Probleme propuse

1. Orologiul

Un orologiu indică ora **h** și **m** minute. Ce oră va indica orologiu peste **k** minute? Câte zile au trecut de la ziua inițială?

2. Divizori

Se introduc de la tastatură două numere naturale **a** și **b**. Să se stabilească dacă **b** este un divizor al lui **a**.

Programul va afișa mesajul "DA" sau "NU".

💡 Tricks & Tips

◆ Dacă lucrați cu numere reale evitați testele de forma $(x1 = x2)$, unde $x1$ și $x2$ sunt numere reale.

Datorită faptului că numerele sunt reprezentate intern printr-un număr finit de biți ($8 * 6 \text{ bytes} = 48 \text{ biți}$ în cazul tipului **real**) este imposibil să se reprezinte întotdeauna cu exactitate o valoare fracționară cu multe zecimale. De asemenea, calculatorul poate să reprezinte, de exemplu, valoarea 0,2 ca 0,1999999.

Datorită reprezentării aproximative a numerelor fracționare este indicat să comparăm valoarea absolută a diferenței numerelor fracționare cu o constantă de valoare foarte mică, $\text{eps} = 0,000001$.

```
program numere_reale_egale;  
const eps = 0.000001;  
var x1, x2:real;  
begin  
    ...  
    if abs(x1 - x2) < eps then  
        writeln('Numere reale egale');  
    ...  
end.
```

◆ Pentru a verifica dacă un număr real **x** este **pătrat perfect** putem verifica dacă partea fracționară a rădăcinii pătrate a numărului este zero.

```
program patrat_perfect;  
var x:real;  
begin  
    write('x ='); readln(x);  
    if (x >= 0) then  
        if frac(sqrt(x)) = 0 then  
            writeln(' este patrat perfect')  
        else writeln(' nu este patrat perfect')  
    else writeln('x nu poate fi negativ');  
end.
```


Stil de programare

Timpul acordat citirii și analizei problemei trebuie să fie mai mare decât timpul necesar scrierii programului.

Alegeți nume sugestive pentru variabile.

Calitatea testelor este mai importantă decât cantitatea lor.

Datele de test trebuie să conțină valori tipice, dar și valori extreme, precum și valori ilegale.

11. Caruselul instrucțiunilor

Una dintre cele mai frecvente operații pe care le efectuează un program este repetarea unui grup de instrucțiuni de un anumit număr de ori, adică realizarea unei **bucle de instrucțiuni**.

O buclă este un subproces al programului în care un grup de acțiuni se repetă succesiv de un număr finit de ori. Numărul finit de repetări (sau iterații) este asigurat de prezența unei **condiții**. Condiția trebuie să conțină cel puțin **un element variabil** care va fi modificat de grupul de instrucțiuni al subprocesului, asigurându-se astfel ieșirea din buclă.

Pentru implementarea algoritmilor repetitivi cu număr cunoscut de iterații sau pași se utilizează instrucțiunea FOR, iar pentru implementarea algoritmilor repetitivi cu condiție se utilizează instrucțiunea WHILE sau instrucțiunea REPEAT.

O paralelă între instrucțiunile repetitive FOR, REPEAT și WHILE

Profesorul de educație fizică al unei școli era foarte preocupat de realizarea unui document privind înălțimile medii ale elevilor școlii, pe categorii de vârstă.

Mihnea, un elev din clasa a VI-a, câștigătorul multor concursuri de programare, s-a oferit să scrie un program care să ușureze mult munca domnului profesor.

Pornind de la formula de calcul a înălțimii medii (suma înălțimilor tuturor elevilor de o anumită vârstă/numărul de elevi de aceea vârstă), Mihnea a construit următorul algoritm:

- P1. Se inițializează suma înălțimilor elevilor (**s**) cu 0.
- P2. Se tastează vârsta dorită (**v**) și numărul elevilor de aceea vârstă (**n**).
- P3. Se tastează înălțimea unui elev și se memorează în variabila (**h**).
- P4. Se adună înălțimea tastată la suma **s**.
- P5. Dacă mai există elevi a căror înălțime nu a fost încă tastată, se reia pasul P3, în caz contrar, se trece la pasul P6.

P6. Se calculează înălțimea medie.

P7. Se afișează înălțimea medie.

Observații

1. Dacă ar fi utilizat câte o variabilă pentru păstrarea înălțimii fiecărui elev, Mihnea ar fi fost „greu încercat” la scrierea acestui program, deoarece ar fi avut nevoie de sute de variabile, care conduceau la sute de instrucțiuni de citire și atribuire, adică la un program tip „spaghetti”.

Să nu uităm că limbajul Pascal a revoluționat lumea programării prin introducerea conceptului de **programare structurată**: orice algoritm poate fi construit utilizând doar trei tipuri de structuri: liniare, condiționale și repetitive.

2. Mihnea a rezolvat această problemă utilizând o structură repetitivă cu număr cunoscut de pași, numărul de pași fiind egal cu numărul de elevi de vârstă specificată. Este posibilă utilizarea unei singure variabile (h) pentru memorarea, pe rând, a înălțimilor elevilor, deoarece după ce s-a adunat la sumă (s), nu mai este nevoie de înălțimea acelui elev și se poate memora în variabila h înălțimea elevului următor.

3. ieșirea din buclă are loc după tastarea înălțimii ultimului elev.

```
program demo_FOR;
var v,n,h,i:integer;
    s:longint;h_med:real;
begin
    { Blocul de inițializare – opțional în Borland Pascal 7.0 }
    s:=0;
    { Citirea datelor de intrare }
    write('Introduceti varsta:');readln(v);
    write('Introduceti numarul de elevi:');readln(n);
    { Bucula de calcul pentru suma înălțimilor tuturor elevilor }
    for i:=1 to n do
    begin
        write('Introduceti inaltimea (cm):');readln(h);
        s:=s+h;
    end;
    { Calculul și afișarea rezultatelor }
    h_med:=s/n;
    write('Inaltimea medie a elevilor de ',v,' ani =',
        h_med:5:2,' cm');
end.
```

Să executăm pas cu pas programul de mai sus pentru următoarele date de intrare:
n = 5; v = 13; h = 155, 160, 158, 164, 160.


```

h_med:=s/n;
write('Inaltimea medie a
      elevilor de ',v,' ani
      = ',h_med:5:2,' cm');
readln;
end.

```

Numărul de repetări, sau de pași, nu este cunoscut din faza de programare, depinzând de momentul în care se va tasta înălțimea de valoare 0.

REPEAT este o instrucțiune repetitivă cu număr necunoscut de pași și test final.

Chiar în momentul în care se pregătea să tasteze înălțimea primului elev, profesorul a fost chemat în cancelaria școlii și atunci a tastat 0 pentru întreruperea programului. Efectul acestei manevre a fost eroarea de execuție **"Division by zero"**.

Anunțat de bug-ul apărut, Mihnea a încercat să afle cauza terminării anormale a programului. A observat că înălțimea „elevului fictiv” fiind 0 nu afectează suma s , dar n devenind 0 în urma regularizării (neexistând alt elev în afara „elevului fictiv”) are loc o împărțire la 0, ceea ce determină eroarea de execuție **"Division by zero"**.

REPEAT fiind o instrucțiune repetitivă cu test final, a înlocuit-o cu o instrucțiune repetitivă cu test inițial, și anume, instrucțiunea WHILE.

Se citește înălțimea primului elev înaintea intrării în bucla WHILE. Dacă prima înălțime tastată este 0, se ocolește bucla de calcul a sumei înălțimilor elevilor.

Se obține următoarea variantă de program fără „bug-uri”:

```

program demo_WHILE;
var   v,n,h,i:integer;
      s:longint;h_med:real;
begin
  s:=0; n:=0;
  write('Introduceti varsta:'); readln(v);
  write('Introduceti inaltimea
        primului elev (cm):'); readln(h);
  while h<>0 do
    begin
      n:=n+1; s:=s+h;
      write('Introduceti inaltimea
            elevului urmator (cm):');
      readln(h);
    end;
  if n<>0 then
    begin
      h_med:=s/n;
      write('Inaltimea medie a elevilor
            de ',v,' ani = ',h_med:5:2,' cm')
    end
  else
    write('Nu exista nici un elev de
          ',v,' ani!'); readln;
  end.

```

Evoluția valorilor variabilelor n , h și s pentru următoarele date de intrare $v=13$; $h=0$.

Prima înălțime tastată fiind 0, condiția instrucțiunii **while** devine FALSĂ și bucla WHILE este evitată, executându-se instrucțiunea următoare instrucțiunii **while**, și anume, instrucțiunea **if $n \neq 0$** ... Deoarece n a rămas cu valoarea inițială 0, programul va afișa mesajul "Nu există nici un elev de ... ani!".

WHILE este o instrucțiune repetitivă cu număr necunoscut de pași și test inițial. Numărul minim de repetări pentru WHILE este 0, iar pentru REPEAT este 1.

Bucle de instrucțiuni

Părțile unei bucle de instrucțiuni sunt date în tabelul 48:

Părțile	Operația	Semnificație
Partea I	Inițializare	Inițializează variabila de control a buclei
Partea II	Testare	Testează variabila de control sau condiția
Partea III	Execuție	Execută grupul de instrucțiuni
Partea IV	Modificare	Modifică valoarea variabilei de control sau efectuează o operație care afectează condiția testată

Tabelul 48

Sintaxa instrucțiunilor repetitive

Instrucțiunea care se repetă poate fi o instrucțiune simplă sau o instrucțiune compusă, delimitată de cuvintele rezervate *begin* și *end*. În cazul instrucțiunii **repeat** *begin* și *end* pot să lipsească, deoarece delimitarea este făcută de cuvintele rezervate **repeat** și **until**.

Tabelul 49

Instrucțiune	Execuție
for variabila_{control} := valoare_{inițială} to valoare_{finală} do instrucțiune; Variabila de control trebuie să fie de tip ordinal, un tip a cărui mulțime de valori este o mulțime ordonată, pe care sunt definite funcțiile <i>succ</i>, <i>pred</i> și <i>ord</i> (tipurile întregi, char, enumerare). Variabila de control nu poate fi de tip real!	Variabila_{control} avansează cu 1 de la valoare_{inițială} la valoare_{finală} • Dacă valoare_{inițială} = valoare_{finală} ⇒ ⇒ numărul de pași = 1. • Dacă valoare_{inițială} > valoare_{finală} ⇒ ⇒ numărul de pași = 0 • Dacă valoare_{inițială} < valoare_{finală} ⇒ ⇒ numărul de pași = valoare_{finală} - valoare_{inițială} + 1
for variabila_{control} := valoare_{inițială} downto valoare_{finală} do instrucțiune;	Variabila_{control} descrește cu 1 de la valoare_{inițială} la valoare_{finală}.
repeat instrucțiune until condiție;	Se repetă instrucțiunea până când condiția devine adevărată. • Număr necunoscut de repetări. • Număr minim de repetări = 1.
while condiție do instrucțiune;	Se repetă instrucțiunea cât timp condiția este adevărată. • Număr necunoscut de repetări. • Număr minim de repetări = 0.

Ieșirea forțată dintr-o buclă

Instrucțiunea BREAK

Instrucțiunea BREAK prezintă în grupul de instrucțiuni care se repetă forțează ieșirea din instrucțiunea repetitivă și continuarea execuției programului cu următoarea instrucțiune după instrucțiunea repetitivă.

Instrucțiunea CONTINUE

Instrucțiunea CONTINUE prezintă în grupul de instrucțiuni care se repetă forțează ieșirea din pasul curent și determină saltul la verificarea condiției instrucțiunii repetitive.

Instrucțiunile EXIT și HALT

Instrucțiunile EXIT și HALT prezente în orice loc din programul principal forțează ieșirea din program.

12. Aplicații cu instrucțiuni repetitive

Mulțimea divizorilor proprii ai unui număr natural

Problema 1. Divizori

Să se afișeze mulțimea divizorilor proprii ai unui număr natural n , introdus de la tastatură ($n < 10000$).

1. Analiza problemei

1.1. Datele problemei

- a) Date de intrare: n :word;
- b) Date de ieșire: mulțimea divizorilor proprii ai lui n

d este divizorul lui n dacă și numai dacă există un număr natural c , $c \neq 0$ astfel încât are loc egalitatea $n = c \cdot d$. Conform teoremei împărțirii cu rest, deducem că n se împarte exact la d , adică restul împărțirii lui n la d este 0. Divizorii proprii sunt divizorii lui n diferiți de 1 și n .

1.2. Funcția programului

Se verifică toate numerele cuprinse între 2 și $(n \div 2)$ dacă sunt divizori ai lui n .

Parcurgerea tuturor numerelor cuprinse între 2 și $(n \div 2)$ se va face cu ajutorul unei instrucțiuni repetitive. Deoarece este cunoscut numărul de pași $((n \div 2) - 2 + 1)$ se va utiliza instrucțiunea FOR.

2. Setul de teste

Pentru a construi un algoritm corect să încercăm întâi să alegem un set de teste (tabelul 50) care să includă valorile extreme și ilegale pentru n . Algoritmul va trebui să țină seama în faza de proiectare de aceste valori extreme și ilegale.

			Tabelul 50
Nr. test	n	Mulțimea divizorilor proprii	Observații
1.	4	$D_4 = \{2\}$	valoare tipică
2.	12	$D_{12} = \{2; 3; 4; 6\}$	valoare tipică
3.	42	$D_{42} = \{2; 3; 6; 7; 14; 21\}$	valoare tipică
4.	1	$D_1 = \emptyset$	valoare tipică
5.	3	$D_3 = \emptyset$	valoare tipică
6.	0	$D_0 = 0$ infinitate de divizori	valoare extremă
7.	10000	Data de intrare incorectă. Se va cere o nouă valoare.	valoare ilegală

3. Implementarea algoritmului în limbajul de programare Pascal

• Varianta cu instrucțiunea FOR

```

program divizori_FOR;
var    n,d:word;
        exista_divizori:boolean; { "variabilă martor": inițializată pe FALSE, va
                                   primi valoarea TRUE în momentul în care se va
                                   găsi primul divizor propriu }

begin
    exista_divizori:=false;
    { Filtrul pentru validarea datelor de intrare }
    REPEAT
        write('Introduceti n (n<10000):');
        readln(n);
    UNTIL n<10000;
    if n=0 then write('0 infinitate de divizori')
    else
        begin
            write('D',n,'={');
            for d:=2 to n div 2 do
                if n mod d=0 then
                    begin
                        write(d,', ');exista_divizori:=true;
                    end;
            write(#8); { #8 este codul tastei <BackSpace>, necesară pentru ștergerea
                        virgulei după ultimul divizor }
        end;

```



```

        if exista_divizori then write('}')
        else write('Multimea vida');
    end; readln;
end.

```

• *Varianta cu instrucțiunea WHILE*

```

program divizori_WHILE;
var  n,d:word; exista_divizori:boolean;
begin
    exista_divizori:=false;
    REPEAT
        write('Introduceti n (n<10000):');
        readln(n);
    UNTIL n<10000;
    if n=0 then write('0 infinitate de divizori')
    else
        begin
            write('D',n,'={');
            d:=2;
            WHILE d<=n div 2 do
            BEGIN
                if n mod d=0 then
                begin
                    write(d,',');exista_divizori:=true;
                end;
                d:=d+1;
            END;
            write('#8');
            if exista_divizori then write('}')
            else write('Multimea vida');
        end; readln;
    end.

```

Observație. Dacă se utilizează instrucțiunea WHILE pentru implementarea unui algoritm cu număr cunoscut de pași, programatorul trebuie să scrie instrucțiunile pentru inițializarea și avansul variabilei de control, operații implicate în cadrul instrucțiunii FOR.

Strămoșul algoritmilor — algoritmul lui Euclid

Problema 2. Tort (100 de puncte)

Un tort dreptunghiular de dimensiuni $M \times N$ trebuie împărțit în porții pătrate de aceeași mărime.

Cerință: Găsiți numărul minim de porții care se pot obține și dimensiunea L a acestora. Atât dimensiunile dreptunghiului cât și ale pătratelor în care se împarte sunt numere întregi.

Date de intrare: Fișierul `tort.in` va conține pe prima linie numerele M și N separate printr-un spațiu.

Date de ieșire: Fișierul `tort.out` va conține, pe o singură linie, două numere naturale separate printr-un spațiu, primul fiind numărul minim de porții, iar celălalt dimensiunea L .

Restricții: $1 \leq M, N \leq 10000$.

Exemplu:

`tort.in` `tort.out`

30 4 20 24

Timp maxim de execuție: 1 secundă/test.

(ONIGIM — Olimpiada Națională de Informatică pentru Gimnaziu
Faza pe Internet, 2003, clasa a VIII-a)

1. Implementarea algoritmului în limbajul de programare Pascal

```
program tort;  
var m,n,r,cmmdc,nr,c_n,c_m:longint; fin,fout:text;  
begin  
    { Deschiderea fișierelor de intrare și ieșire }  
    assign(fin,'tort.in'); reset(fin);  
    assign(fout,'tort.out'); rewrite(fout);  
    { Citirea datelor de intrare din fișierul de intrare cu identificatorul intern fin }  
    readln(fin,m,n);  
    c_m:=m; c_n:=n; { copiile lui m și n }  
    { Calculul cmmdc(m,n) cu algoritmul lui Euclid }  
    while n<>0 do  
        begin  
            r:=m mod n; m:=n; n:=r  
        end;  
    cmmdc:=m;  
    nr := (c_m div cmmdc)*(c_n div cmmdc);  
    { Scrierea datelor de ieșire în fișierul text cu identificatorul intern fout }  
    writeln(fout,nr,' ',cmmdc);  
    { Închiderea fișierelor text }  
    close(fin); close(fout); end.
```


2. Testarea programului

Tabelul 51

Nr. test	tort.in	tort.out	Nr. test	tort.in	tort.out
1.	12 15	20 3	6.	1200 1800	6 600
2.	44 33	12 11	7.	1000 1001	1001000 1
3.	144 133	19152 1	8.	3745 3531	1155 107
4.	83 83	1 83	9.	7600 5415	4560 95
5.	200 40	5 40	10.	8872 9444	5236698 4

Numărul de divizori ai unui număr natural dat

Problema 3. Ușile

n uși sunt inițial descuiate, existând o cheie pentru fiecare ușă. n persoane primesc câte o cheie și au de făcut următorul lucru: persoana cu numărul de ordine egal cu i se va duce pe rând la ușile cu numerele $i, 2i, 3i, \dots$, și va încuia ușa descuiată sau va descuia ușa încuiată. După ce toate cele n persoane au acționat în modul descris mai sus, respectând ordinea persoanelor, câte uși rămân încuiate și care sunt acestea?

n se introduce de la tastatură și $1 \leq n \leq 1000000$.

1. Analiza problemei

Ca să putem construi algoritmul unei probleme este util, de foarte multe ori, să rezolvăm problema pentru un caz particular, să analizăm pașii soluției cazului particular și să descoperim „cheia” generalizării soluției.

Rezolvăm problema pentru $n=13$. Datele de intrare și pașii parcurși sunt ilustrați în tabelul 52.

Tabelul 52

ușa persoana	1	2	3	4	5	6	7	8	9	10	11	12	13
1	x	x	x	x	x	x	x	x	x	x	x	x	x
2		x		x		x		x		x		x	x
3			x			x			x			x	
4				x				x				x	
5					x							x	
6						x				x			
7							x				x		
8								x					
9									x				
10										x			
11											x		
12												x	
13													x
Nr. total de acționări	1	2	2	3	2	4	2	4	3	4	2	6	2
Uși încuiate	da			da					da				

Ce putem observa din acest exemplu?

- Deoarece inițial toate ușile erau descuiate, numai ușile acționate de un număr impar de ori vor rămâne, în final, încuiate.
- Fiecare ușă este acționată de un număr de ori egal cu numărul de divizori ai numărului de ordine al ușii.

Conform celor două observații va trebui să determinăm acele numere naturale care au un număr impar de divizori. Vom apela la cunoștințele de matematică, ținând seama de faptul că „întreaga noastră viață este impregnată de matematică” (Paul Montel).

Fie descompunerea în factori primi a numărului natural x : $x = f_1^{p_1} \cdot f_2^{p_2} \cdot \dots \cdot f_m^{p_m}$

Numărul total de divizori ai numărului natural x : $nd = (p_1+1) \cdot (p_2+1) \cdot \dots \cdot (p_m+1)$

Dacă x este pătrat perfect atunci toți factorii primi din descompunerea lui x vor apărea la o putere pară. În acest caz, sumele (p_1+1) , (p_2+1) , ..., (p_m+1) vor fi impare, iar produsul $(p_1+1) \cdot (p_2+1) \cdot \dots \cdot (p_m+1)$ va fi impar.

Dacă x nu este pătrat perfect el va avea cel puțin un factor prim care va apărea la o putere impară. În acest caz, va exista cel puțin un factor (p_i+1) par ceea ce va determina paritatea numărului de divizori.

Orice pătrat perfect are un număr impar de divizori.

Orice număr care nu este pătrat perfect are un număr par de divizori.

2. Funcția programului

Se determină toate pătratele perfecte mai mici sau egale cu n . Numărul de uși încuiate este egal cu partea întreagă a lui $\sqrt{n}$.

3. Implementarea algoritmului în limbajul de programare Pascal

```
program usable;  
var n,i:longint;  
begin  
    write('Cate uși sunt?');readln(n);  
    writeln(int(sqrt(n)):7:0,' uși încuiate.');
```

write('Ușile încuiate:');

```
    for i:=1 to n do  
        if frac(sqrt(i))=0 then write(i,' ');  
    readln;  
end.
```


Cifrele unui număr natural

Problema 4. Castane

Într-un parc cu castani un copil găsește n castane. Copilul se joacă cu cifrele numărului de castane, conform următoarelor exemple:

$173 \rightarrow 11 \rightarrow 2$

$8 \rightarrow 8$

$1989 \rightarrow 27 \rightarrow 9$

Cerință: Ce cifră va obține copilul pentru numărul de castane culese?

Date de intrare: Numărul de castane se introduce de la tastatură.

Restricții: $n \leq 2\,000\,000\,000$. Datele de intrare se consideră corecte.

(Concursul Național de Informatică al Cluburilor și Palatelor Copiilor²
Satu-Mare, octombrie 2002, clasa a VI-a)

1. Analiza problemei

Este o problemă care face apel la spiritul de observație al concurenților.

Se observă că $1 + 7 + 3 = 11$ și $1 + 1 = 2$ sau $1 + 9 + 8 + 9 = 27$ și $2 + 7 = 9$.

2. Descrierea algoritmului

P1. Se citește n .

P2. Se extrag cifrele numărului n și se însumează.

P3. Dacă suma obținută este formată dintr-o singură cifră se trece la pasul **P4**, în caz contrar, se transferă suma în n și se reia pasul **P2**.

P4. Se afișează numărul format dintr-o singură cifră.

3. Implementarea algoritmului în limbajul de programare Pascal

program **castane**;

var n :longint; s :integer;

 cifra:byte;

begin

 write('n=');readln(n);

 REPEAT

$s:=0$;

 REPEAT

$cifra:=n \bmod 10$;

$s:=s+cifra$;

$n:=n \div 10$;

 UNTIL $n=0$;

$n:=s$;

 UNTIL $s \leq 9$;

 write(s);

readln; end.

Extragerea cifrelor unui număr scris în baza 10 se face prin împărțiri succesive la 10. Prima cifră extrasă va fi cea mai puțin semnificativă cifră a numărului.

n	$n \div 10$	$n \bmod 10$	
$173 : 10 =$	17	rest 3	
$17 : 10 =$	1	rest 7	
$1 : 10 =$	0	rest 1	
0			cifrele numărului

² Concursul Național de Informatică al Cluburilor și Palatelor Copiilor este organizat, an de an, începând cu anul 1990, de către Palatul Copiilor și Inspectoratul Școlar Județean din câte un județ al țării. Fiecare județ participă cu câte un echipaj format din 4 elevi din clasele V-VIII.

Codificare

Problema 5. Baze ascunse

Se dau următoarele corespondențe pentru numerele naturale:

0	corespunde	0
1	corespunde	143
2	corespunde	307
3	corespunde	453
7	corespunde	1273
11	corespunde	2113
576	corespunde	160377

Să se determine codificarea numerelor citite din fișierul cu numele **cni.in** (fiecare număr fiind citit de pe o linie a fișierului). Datele de ieșire se vor scrie în fișierul **cni.out**. **Restricții:** Numerele din fișierul de intrare sunt numere naturale ≤ 999999 .

(Concursul Național de Informatică al Cluburilor și Palatelor Copiilor
Satu-Mare, octombrie 2002, clasa a VIII-a)

1. Analiza problemei

Este o problemă care face apel la ingeniozitatea concurenților.

Se poate observa că toate codurile sunt formate din cifrele 0..7. Presupunând că numerele din coloana (2) sunt scrise în baza 8, vom converti numerele în baza 10 și apoi vom analiza corespondențele dintre coloanele (1) și (3) (tabelul 53).

Tabelul 53

Coloana 1 din enunț	Coloana 2 din enunț	Coloana 2 convertită în baza 10
(1)	(2)	(3)
0	0	0
1	143	99
2	307	199
3	453	299
7	1273	699
10	1747	999
11	2113	1099
576	160377	57599

Se observă că fiecare cod din coloana (3) din tabelul 55 se obține din concatenarea numărului corespunzător din coloana (1), micșorat cu 1 și grupul de cifre 99.

Programul va trebui să urmeze traseul invers raționamentului de mai sus, și anume, pornind de la numărul din coloana (1), va construi codul din coloana (3) și apoi îl va converti în baza 8, obținând codul din coloana (2).

2. Implementarea algoritmului în limbajul de programare Pascal

```
program baze_ascunse;
var n,cod,cod_baza_8,nr_inv: longint; cifra:byte; fin, fout:
text;
begin
    assign(fin, 'cni.in'); reset(fin);
    assign(fout, 'cni.out'); rewrite(fout);
    while not eof(fin) do
        begin
            readln(fin, n);
            if n = 0 then
                writeln(fout,0)
            else
                begin {Obținerea codului din coloana 3}
                    cod := n - 1;
                    cod := cod * 10 + 9; cod := cod * 10 + 9;
                    {Conversia în baza 8}
                    cod_baza_8:=0;
                    repeat
                        cifra:=cod mod 8; {Prima cifră extrasă este cea mai puțin
                                            semnificativă}
                        cod_baza_8:=cod_baza_8*10+cifra;
                        cod:=cod div 8;
                    until cod=0;
                    nr_inv:=0; {Inversarea codului anterior obținut}
                    repeat
                        cifra:=cod_baza_8 mod 10;
                        nr_inv:=nr_inv*10+cifra;
                        cod_baza_8:=cod_baza_8 div 10;
                    until cod_baza_8=0;
                    {Scrierea rezultatului în fișierul de ieșire}
                    writeln(fout,nr_inv);
                end;
            end; close(fin); close(fout);end.
```

Termenul general al unei sume

Fie suma $S = 1 + 2 + 3 + 4 + 5 + \dots$

Prin **termenul de rang i** se înțelege al i-lea termen din sumă.

Ce formă va avea T_i din suma S?

Răspuns: $T_i = i$.

Cunoscând expresia termenului de rang i al sumei putem calcula suma primilor n termeni ai unei sume, astfel:

1. Descrierea algoritmului

```
inceput calcul_suma
citește n
S ← 0
pentru i ← 1 la n execută
    Ti ← i;
    S ← S + Ti
scrie S
sfârșit calcul_suma
```

2. Implementarea algoritmului în limbajul de programare Pascal

```
program calcul_suma;
var n, S, i, Ti: longint;
begin
    S := 0;
    write('n='); readln(n);
    for i := 1 to n do
    begin
        Ti := i;
        S := S + Ti;
    end;
    write('S=', S); readln; end.
```

Observație. Suma primelor n numere naturale se poate calcula utilizând formula matematică:

$$1 + 2 + 3 + 4 + 5 + \dots + n = n(n + 1)/2$$

Formule utile

- Fie suma $S = 1 + 3 + 5 + 7 + \dots$

$$\Rightarrow T_i = 2i - 1$$

Formula de calcul al sumei primelor n numere impare: $1 + 3 + 5 + 7 + \dots + (2n - 1) = n^2$.

- Fie suma $S = 2 + 4 + 6 + 8 + \dots$

$$\Rightarrow T_i = 2i$$

Formula de calcul al sumei primelor n numere pare: $2 + 4 + 6 + 8 + \dots + (2n) = n(n + 1)$.

13. Numere magice

„Matematica pură este o cheie de aur care deschide toate științele.”
Victor Duruy

Numerele lui Nicomachos

Problema grupării numerelor impare a fost formulată de matematicianul Nicomachos în cartea sa „Arithmetike eisagoge” în anul 100 d.Hr.

Problema 7. Numere Nicomachos

Observăm că:

$$\begin{aligned} 1^3 &= 1 \\ 2^3 &= 3 + 5 \\ 3^3 &= 7 + 9 + 11 \end{aligned}$$

$$4^3 = 13 + 15 + 17 + 19$$

.....

Să se scrie n^3 ca sumă de numere impare. Numărul n se citește din fișierul de intrare `nicho.in`, iar descompunerea lui n^3 se va scrie în fișierul `nicho.out` (câte un termen pe rând). Restricții:
 $n \leq 1\,000\,000$.

1. Analiza problemei

Numerele naturale impare se grupează astfel:

$$1 + (3 + 5) + (7 + 9 + 11) + (13 + 15 + 17 + 19) + \dots$$

Grupa 1 are 1 termen.

Grupa 2 are 2 termeni.

...

Grupa $(n - 1)$ are $(n - 1)$ termeni.

- Primele $(n - 1)$ grupe vor fi formate din $1 + 2 + 3 + \dots + (n - 1) = n(n - 1)/2$ numere impare.
- Ultimul număr din grupa cu numărul de ordine $(n - 1)$ va avea rangul $n(n - 1)/2$.
- Primul număr din grupa cu numărul de ordine n va avea rangul $n(n - 1)/2 + 1$.
- Termenul general al șirului de numere impare fiind $T_i = 2i - 1 \Rightarrow$ a n -a grupă va începe cu termenul $2[n(n - 1)/2 + 1] - 1$.

2. Implementarea algoritmului în limbajul de programare Pascal

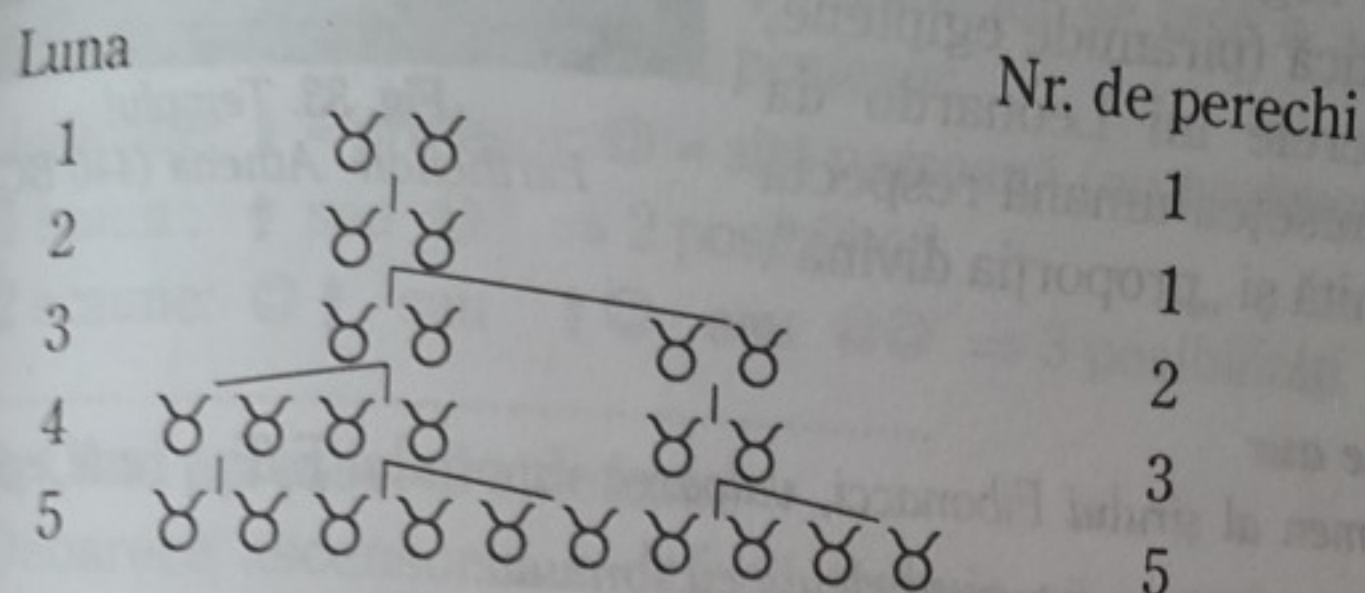
```
program numere_Nicomachos;
var n,i,ti:longint; fi,fo:text;
begin
    assign(fi,'nicho.in');reset(fi);
    assign(fo,'nicho.out');rewrite(fo);
    readln(fi,n);
    ti:=2*(n*(n-1) div 2+1)-1; {Primul termen din a n-a grupă}
    for i:=1 to n do
    begin
        writeln(fo,ti);
        ti:=ti+2; {În a n-a grupă sunt n termeni impari}
    end;
    close(fi);close(fo);
end.
```

Șirul Fibonacci

Leonardo Fibonacci a fost un matematician italian care a trăit în Pisa în perioada 1170–1240. În opera sa „Liber abaci”, el a sistematizat teoreme ale antichității și a prezentat propriile metode matematice.

Problema 8. Pereche de iepuri (prezentată în cartea „Liber abaci”)

O pereche de iepuri nou-născuți este pusă într-un câmp. La sfârșitul celei de-a doua luni de viață prima pereche de iepuri produce altă pereche de iepuri. Presupunând că iepurii nu mor nicio-
 dată și că fiecare pereche produce o nouă pereche în fiecare lună, începând din a doua lună de
 viață, câte perechi de iepuri există pe câmp, după n luni?



Numărul perechilor de iepuri formează șirul lui Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, ... în care fiecare termen se obține însumând cei doi termeni anteriori $2 = 1 + 1$, $3 = 1 + 2$, $5 = 2 + 3$, ...

Termenul de rang i al șirului se obține cu ajutorul formulei de recurență:
 $F_i = F_{i-1} + F_{i-2}$.

1. Implementarea algoritmului în limbajul de programare Pascal

```
program perechi_de_iepuri;
```

```
var n, l: integer;
```

```
    la, lp, lc: longint; {lc = numărul perechilor de iepuri din luna curentă;  
                        lp = numărul perechilor de iepuri din luna precedentă  
                        la = numărul perechilor de iepuri din luna  
                        anteprecedentă}
```

```
begin
```

```
    write('Cate luni:'); readln(n);
```

```
    if (n=1) or (n=2) then lc:=1
```

```
    else
```

```
    begin
```

```
        la:=1; lp:=1;
```

```
        for l:=3 to n do
```

```
        begin
```

```
            lc:=la+lp; {Formula de recurență}
```

```
            la:=lp; {La trecerea la luna următoare se face rocada:}
```

```
            lp:=lc; {la ← ① lp ← ②lc}
```

```
        end;
```

```
    end;
```

```
    writeln('Dupa ', n, ' luni exista ', lc, ' perechi de iepuri.  
            ');
```

```
end.
```


Numărul de aur

Numărul de aur sau proporția de aur (the golden number) este $\phi = (1 + \sqrt{5})/2 = 1,6180339887498949$ și se regăsește în natură, în arhitectura modernă și antică (piramide egiptene, templele grecești), în operele lui Leonardo da Vinci, Michelangelo. Frumusețea umană respectă proporția de aur, supranumită și „proporția divină”.

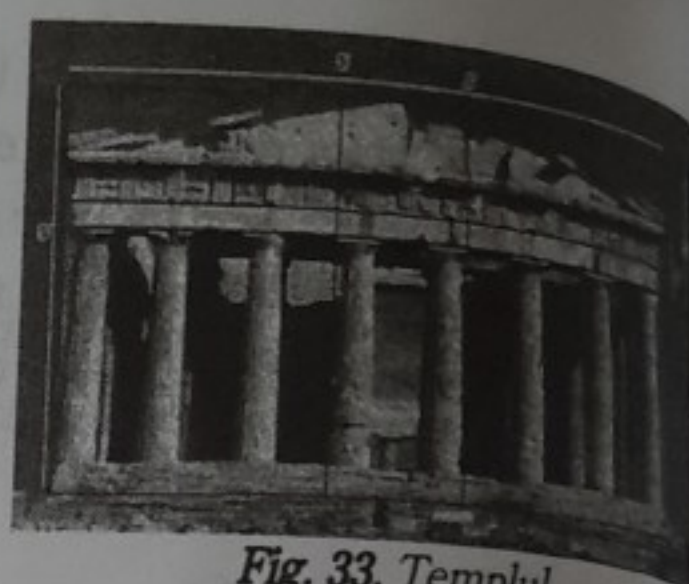


Fig. 33. Templul Parthenon - Athena (440 BC)

Problema 9. Numărul de aur

Începând cu al 41-lea termen al șirului Fibonacci, valoarea raportului F_i/F_{i-1} este egală cu numărul de aur ϕ . Termenul de rang i se va putea calcula cu formula:

$$F_i = \text{round}(\phi^i / \sqrt{5}).$$

1. Implementarea algoritmului în limbajul de programare Pascal

```
program numarul_de_aur;
{$N+}
var n, i, j, Fi:longint; phi,p:extended;
begin
    write('n='); readln(n);
    phi:=(1+sqrt(5))/2; {phi este notația pentru φ}
    {Calculul termenilor Fibonacci, utilizând numărul de aur}
    for i:=1 to n do
        begin
            p:=1; {Calculul lui φ^i}
            for j:=1 to i do p:=p*phi;
            Fi:=round(p/sqrt(5));
            write(Fi, ' ');
        end;
    end.
```

Deoarece valorile termenilor Fibonacci cresc foarte repede, ultimul termen pe care îl putem obține, utilizând tipul *longint*, este termenul de rang 46.

Pentru a obține cu precizie termeni Fibonacci de rang mare putem utiliza „numere uriașe”, reprezentate cu ajutorul tablourilor unidimensionale (lecția 16 din partea a II-a).

Puzzles cu numere Fibonacci

Șirul Fibonacci se regăsește în natură (numărul de petale sau numărul de crengi ale unei plante, organizarea coloniilor de albine, spirala cochiliei scoicilor), în compoziții muzicale, în poezii.

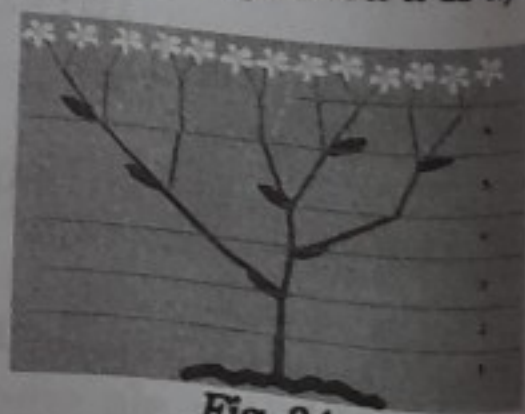


Fig. 34. Șirul lui Fibonacci în natură

Probleme propuse

1. Scaune pe un singur rând fără ... profesori vecini

Pe un rând format din n scaune trebuie să ia loc n persoane, dintre care o parte sunt profesori. Deoarece doi vecini profesori ar vorbi mult despre școală, aflați toate modalitățile de așezare a persoanelor pe cele n scaune, astfel încât nici un profesor să nu aibă ca vecin alt profesor.

Indicație: ♣ = profesor; ☺ = altă persoană (nu profesor)

1 scaun : ♣ sau ☺ $\Rightarrow$ 2 posibilități

2 scaune: ☺ ♣ sau ♣ ☺ sau ☺ ☺ $\Rightarrow$ 3 posibilități

2. Când ascensorul e defect...

Deoarece ascensorul e defect, trebuie să urcați singuri cele n scări. Fiind grăbiți, puteți urca scară cu scară, sau câte două scări o dată. Câte posibilități există pentru a urca cele n scări?

14. Produsul primelor n numere naturale

Problema 10. Factorial (100 puncte)

Se dă un număr natural n ($n \leq 1000$) și un număr natural prim p .

Se cere să se afișeze exponentul lui p în descompunerea în factori primi a numărului:

$1 * 2 * 3 * \dots * n$.

Exemplu: $n = 7$ $p = 3$ se va afișa 2

Timp de execuție: 1 secundă/test

(InfoKids 2002³ – Concurs Național de Informatică pentru clasele I–IX, clasa a VI-a)

1. Analiza problemei

$n! = 1 * 2 * 3 * 4 * 5 * 6 * \dots * n$

Dacă am calcula produsul $1 * 2 * 3 * \dots * n$, la $n = 13$ am depăși domeniul de valori al tipului *longint*. Dacă am utiliza tipul *extended* am obține un produs cu maxim 18 cifre exacte, dar $n!$ crește foarte repede; de exemplu, $1000!$ este un întreg cu peste 2 500 de cifre. Va trebui să găsim o soluție care să nu necesite calculul produsului.

- **Soluția 1.** Putem determina la ce putere apare numărul prim p în fiecare factor al factorialului și să însumăm puterile.

³ **InfoKids** este un Concurs Național de Informatică pentru clasele I–IX, inițiat de CYGNUS COMPUTER și organizat în colaborare cu Școala nr. 17 „Pia Brătianu” București (edițiile 2000 – 2001 – 2002) și cu Colegiul Național „Mihai Viteazul” București (ediția 2003). Concursul InfoKids are și alte secțiuni în afară de programarea calculatoarelor (algoritmi): programarea jocurilor pe calculator, programare în modul grafic, crearea paginilor Web, artă pe calculator (crearea poveștilor și grafică).


```

program factorial;
var n,p,e,i,ci:integer;
begin
  readln(n,p);
  e:=0;
  for i:=1 to n do
  begin
    ci:=i; {ci = copia lui i, deoarece variabila de control nu poate fi modificată}
    while ci mod p=0 do {La ce putere apare p în ci?}
    begin
      e:=e+1; ci:=ci div p;
    end;
  end; writeln(e); end.

```

- **Soluția 2.** Numărul prim p este un divizor al lui $n!$ cu *multiplicitatea* egală cu suma $[n/p] + [n/p^2] + [n/p^3] + \dots$, până când $[n/p^i]$ devine 0, $[n/p^i]$ fiind partea întreagă a câtului n/p^i .

Dacă $n = 100$, $p = 3$ va fi un divizor al lui $100!$ cu multiplicitatea: $[100/3] + [100/9] + [100/27] + [100/81] = 33 + 11 + 3 + 1 = 48$ (următorul termen ar fi fost $[100/243] = 0$).

2. Implementarea algoritmului în limbajul de programare Pascal

```

program factorial;
var n,p,e,i,ci,f:integer;
begin
  readln(n,p);
  e:=0; f:=p; {e = multiplicitatea lui p în n!}
  repeat
    e:=e+n div f; {Calcul lui [n/p^i]}
    f:=f*p;
  until n div f=0;
  writeln(e);
end.

```

Teste			
6	3	$\Rightarrow$	2
4	2	$\Rightarrow$	3
11	3	$\Rightarrow$	4
23	5	$\Rightarrow$	4
100	5	$\Rightarrow$	24
211	11	$\Rightarrow$	20
590	23	$\Rightarrow$	26
761	283	$\Rightarrow$	2
831	19	$\Rightarrow$	45
911	137	$\Rightarrow$	6

Probleme propuse

1. Progrămel Softescu

Se citesc de la tastatură două numere naturale n și k . Prietenul nostru, Progrămel Softescu, trebuie să decidă câte cifre finale de zero (aflate la sfârșitul numărului) se obțin prin înmulțirea tuturor numerelor naturale nenule divizibile cu k , numere naturale mai mici sau egale cu n . Problema lui Progrămel Softescu este că n este foarte mare și k este destul de mic, astfel încât există multe numere divizibile cu k . „Îmbătrânesc până calculez un produs cu atâția factori! Și ce factori!”

Sarcina voastră este să scrieți un program care să afișeze pe ecran numărul de zerouri cu care se termină produsul respectiv.

Probleme propuse

Restricții: n are cel mult 8 cifre, iar k are cel mult două cifre.

Exemplu: Dacă luăm $n = 50$ și $k = 3$, atunci produsul numerelor divizibile cu 3 este $3 * 6 * 9 * 12 * 15 * 18 * 21 * 24 * 27 * 30 * 33 * 36 * 39 * 42 * 45 * 48$ și are 3 zerouri la sfârșit.

(ONIGIM 2002 – faza pe municipiu, clasa a VI-a)

Indicații: Pentru a afla numărul de zerouri în care se termină un produs, trebuie să determinăm la ce putere apare 2 în fiecare factor al produsului și să însumăm exponenții. Procedura este identică pentru 5. Minimul dintre exponenții lui 2 și 5 reprezintă numărul de zerouri în care termină produsul.

2. Ultima cifră nenulă

Care este ultima cifră nenulă a produsului $1 * 2 * 3 * 4 * 5 * \dots * n$, n fiind un număr format din cel mult 9 cifre?

Indicații: Pentru a afla ultima cifră diferită de zero a produsului vom determina ultima cifră nenulă a fiecărui factor după ce a fost împărțit la 2, și apoi la 5, cât timp factorul era divizibil cu 2, respectiv 5.

3. Numere „amabile” (semnalate de Abu’Ali ibn al Haisam în anul 1000)

Un număr este amabil dacă este divizibil cu 7 și prin împărțire la 6, 5, 4, 3 și 2 dă restul 1.

Numărul se numește „amabil” deoarece luând în considerare faptul că restul împărțirii numărului la 7 este 0, în descrierea sa apar toate numerele de la 0 la 7.

Să se afișeze toate numerele „amabile” mai mici decât n , $n \leq 2\,000\,000\,000$.

💡 Tips and Tricks

1. Dacă ați uitat sintaxa unei instrucțiuni, poziționați cursorul pe numele instrucțiunii și apăsați simultan tastele CTRL și F1. Veți fi „purtați” direct în Help-ul mediului Pascal la informația dorită.
2. Dacă doriți să executați pas cu pas un program începând cu o anumită instrucțiune, creați un punct de întrerupere (CTRL+F8) la instrucțiunea anterioară instrucțiunii de unde doriți execuția pas cu pas. La lansarea în execuție a programului cu CTRL+F9 sau meniul Run programul se va executa până la punctul de întrerupere, mai departe putând executa programul pas cu pas, prin apăsarea tastei F7. Anularea punctului de întrerupere se face tot prin acționarea simultană a tastelor CTRL și F8.
3. Eficiența unui algoritm se măsoară prin timpul de execuție a programului și spațiul de memorie utilizat. Instrucțiunile repetitive sunt mari „consumatoare de timp” și, de multe ori, timpul necesar execuției unui test fiind mai mare decât cel specificat în enunțul unei probleme aduce concurentului 0 puncte, cu toate că rezultatele testului sunt corecte.

Exemplu: Determinarea timpului de execuție a programului factorial (problema 10 de la pagina 159.)

La adresa absolută de memorie 0000:046C se contorizează numărul de apeluri ale timer-ului, începând de la miezul nopții. Timer-ul este o rutină sistem care se apelează automat la fiecare 55 de milisecunde, aproximativ de 18,2 ori pe secundă.

Pentru a „măsura” timpul de execuție a unui program, memorăm într-o variabilă (**start**) conținutul locației **time** la începutul execuției programului și în altă variabilă (**stop**) conținutul locației **time** la sfârșitul execuției programului.

Durata execuției programului va fi: **(stop-start) / 18,2** secunde.

Calculând timpul consumat de program la fiecare trecere printr-o buclă anumită de instrucțiuni putem crea o „alarmă” care să anunțe depășirea timpului. Se poate întrerupe bucla cu instrucțiunea **break**, sau chiar programul utilizând instrucțiunea **halt**.

```
program factorial;
var n,p,e,i,ci:longint;
    start,stop:longint; fi,fo:text;
time:longint absolute $0000:$046C;
    {$ - adresa este scrisă în baza 16}
begin
    start:=time;
    assign(fi,'fact.in'); reset(fi);
    assign(fo,'fact.out'); rewrite(fi);
    readln(fi,n,p);
    e:=0;
    for i:=1 to n do
        begin
            ci:=i;
            while ci mod p=0 do
                begin
                    e:=e+1;
                    ci:=ci div p;
                end;
            end;
        writeln(fo,e);
        stop:=time;
        writeln('Timp/test=',
            (stop-start) /18.2 :5:2);
    end.
```

Stil de programare

Coloana vertebrală a unui program — pentru ca un program să fie ușor de citit de către alți utilizatori, dar și de către autor, după un anumit interval de timp, este indicată scrierea indentată, astfel încât instrucțiunile de același nivel să fie scrise începând cu același număr de coloană.

Creați o documentație a programului prin utilizarea comentariilor.

Nu sacrificați claritatea programului pentru artificii care ar mări prea puțin eficiența algoritmului.

Nu creați arbori „if” acolo unde puteți utiliza operatori logici în scrierea condițiilor. Utilizați paranteze în expresiile ambigue.

Programați defensiv! Apărați programul în toate „punctele slabe” prin care s-ar putea strecura erori.

Nu vă opriți la prima formă a programului. *Optimizați-o!*

Cele mai multe programe consumă 90% din timp executând doar 10% din instrucțiunile programului. *Arta de a programa înseamnă să găsim aceste 10% instrucțiuni.*

15. Arhitectură cu ... bytes

Tablouri unidimensionale (vectori)

Profesorul de educație fizică al școlii trebuie să realizeze o situație care să conțină valoarea înălțimii maxime și numărul de elevi care au înălțimea egală cu înălțimea maximă, pentru fiecare categorie de vârstă.

Mihnea a scris repede programul de determinare a înălțimii maxime pentru o anumită vârstă, generalizând algoritmul de maxim descris în lecția 10 (problema 3, pagina 126).

Determinarea înălțimii maxime. Se inițializează *max* cu cea mai mică valoare posibilă (0) și se compară fiecare înălțime tastată cu *max* determinat până în acel moment; dacă înălțimea este mai mare decât *max*, *max* își va schimba conținutul, memorând ultima înălțime citită.

```
program inaltimea_maxima;
var n,i:integer;v,h,max:byte;
begin
  write('Introduceti varsta:');readln(v);
  write('Introduceti numarul de elevi:');readln(n);
  max:=0;
  for i:=1 to n do
    begin
      write('Introduceti inaltimea (cm):');readln(h);
      if h>max then max:=h;
    end;
  write('Inaltimea maxima a elevilor de ',v,' ani = ',max,' cm');end.
```

Observații

1. Pentru a determina maximul unor valori întregi (tip integer) variabila *max* se poate inițializa cu (- **MaxInt**), iar pentru valori de tip longint cu (- **MaxLongInt**).
2. Pentru a determina minimul unor valori întregi (tip integer) variabila *min* se va inițializa cu **MaxInt**, iar pentru valori de tip longint cu **MaxLongInt**.

Pentru a calcula câți elevi au înălțimea egală cu înălțimea maximă, trebuie să comparăm înălțimea fiecărui elev cu *max* dat. Variabila *h* conține doar ultima înălțime tastată la sfârșitul procesului de determinare a înălțimii maxime.

Pentru a rezolva această problemă va trebui să păstrăm în memorie înălțimile tuturor elevilor, deci va trebui să găsim un alt mod de organizare a datelor problemei.

Vom memora toate înălțimile tastate într-o zonă contiguă de memorie (formată din locații succesive), în fiecare locație păstrându-se înălțimea unui singur elev. Fiecare locație va avea un număr de ordine în cadrul zonei de memorie, zonă pe care o vom numi h .

Această structură de date se numește **tablou unidimensional** sau **vector** și are următoarele caracteristici:

- toate elementele sunt de același tip, numit *tip de bază*;
- $h[i]$ este elementul de rang (indice) i al tabloului și poate fi prelucrat individual

155	160	158		159		160	
$h[1]$	$h[2]$	$h[3]$	...	$h[i]$	...	$h[n]$	

Pentru a construi *structura de date* necesară rezolvării problemei, Mihnea va trebui să procedeze ca un adevărat *arhitect*, ținând seama de următoarele cerințe:

1. Care este numărul maxim de locații („camere de memorie”), necesar pentru memorarea înălțimilor elevilor din orice categorie de vârstă? (**dim**)
2. Cât de mare trebuie să fie o locație pentru a putea memora orice înălțime posibilă? (**tipul de bază**)
3. Cât de mare este „terenul de memorie” disponibil, pentru a nu ajunge la situații de depășire? Segmentul de date fiind limitat la 64 kB, adică 65536 bytes, trebuie ca $(dim * sizeof(tip\_de\_bază) + dimensiunea\ celorlalte\ variabile\ din\ secțiunea\ var) \leq 65536\ bytes$.

Ținând cont de cerințele de mai sus a fost construit tipul utilizator, **tip_h**. Descrierea unui tip utilizator se face în secțiunea **type**. Constanta **dim** reprezintă *numărul maxim* de elemente (elevi) necesar execuției programului pentru orice categorie de vârstă.

n reprezintă *numărul efectiv* de elemente (elevi) pentru categoria de vârstă pentru care se execută programul, $n \leq dim$.

Înălțimea unui elev fiind mai mică decât 255, **tipul de bază** poate fi tipul byte, integer, word. Pentru a nu „risipi” memoria a fost ales cel mai mic tip posibil, tipul byte. Pentru variabila h de tipul **tip_h** se va alocă următorul spațiu de memorie:

$dim * sizeof(tip\_de\_bază) = 300 * 1\ byte = 300\ bytes$.

Dacă tipul de bază ar fi fost integer, variabila h ar fi ocupat în memoria internă $300 * 2 = 600\ bytes$.

```
program demo_structuri_de_date;
const dim=300;
type tip_h=array[1..dim] of byte;
var n,i,num:integer;v,max:byte; h:tip_h;
begin
  write('Introduceti varsta:');readln(v);
  write('Introduceti numarul de elevi:');readln(n);
```



```

writeln('Introduceti inaltimele elevilor exprimate in cm:');
for i:=1 to n do {Înălțimile celor n elevi vor fi introduse în ordinea tastării în ele-
                 mente succesive ale tabloului h. [și] va memora înălțimea elevului
                 de rang i.}

begin
    write('h[' , i , ']='); readln(h[i]);
end;

    {Determinarea înălțimii maxime}
max:=h[1]; {Se presupune că primul elev este cel mai înalt, max memorând
           înălțimea primului elev.}
for i:=2 to n do
if h[i]>max then max:=h[i]; {Începând cu al doilea elev se compară înălțimea
                           elevului de rang i cu max determinat până la elevul de rang i-1, inclusiv}
                           {Câți elevi au înălțimea maximă?}
num:=0;
for i:=1 to n do
    if h[i]=max then num:=num+1;
writeln('Inaltimea maxima a elevilor de ', v, ' ani = ', max, '
        cm');
writeln(num, ' elevi cu inaltimea maxima. ');
readln;
end.

```

Verificarea unei proprietăți a elementelor vectorului

Problema 1. Să se verifice dacă toți elevii de o anumită vârstă au înălțimea mai mare decât o valoare dată (lim). Programul va afișa mesajul „DA” sau „NU”.

1. Descrierea algoritmului

Început

citește v, lim, n;

pentru $i \leftarrow 1, n$

 citește h[i];

p ← TRUE

poziționare la începutul vectorului $i \leftarrow 1$;

cât timp p = TRUE și nu au fost „vizitate” toate elementele vectorului

 dacă $h[i] \leq \text{lim}$

 atunci p ← FALSE

 altfel avansează la elementul următor

 dacă p = true

 atunci scrie „DA”

 altfel scrie „NU”

sfârșit

2. Implementarea algoritmului în limbajul de programare Pascal

```
program demo_verificare_proprietate;
const dim=300;
type
  tip_h=array[1..dim] of byte;
var n,i,lim:integer;v:byte; p:boolean; h:tip_h;
begin
  write('Introduceti varsta:');readln(v);
  write('Introduceti limita:');readln(lim);
  write('Introduceti numarul de elevi:');readln(n);
  writeln('Introduceti inaltimele elevilor exprimate in cm:');
  for i:=1 to n do
    begin
      write('h[' ,i,']=');readln(h[i]);
    end;
  p:=true; {Inițial p este true, deoarece, până în acest moment, nu am găsit nici un ele-
            ment care să nu verifice proprietatea dată.}
  i:=1; {Poziționare la începutul vectorului}
  while p and (i<=n) do
    if h[i]<=lim then p:=false {Primul element care nu verifică proprietatea
                                poziționează p←FALSE și determină ieșirea din bucla while.}
    else inc(i); {Avans la următorul element al vectorului}
  if p then write('DA')
  else write('NU');
  readln; end.
```

Instrucțiunea repetitivă WHILE are două porți de ieșire din buclă. Dacă s-a ieșit pe poarta „p”, înseamnă că s-a găsit un element care nu respectă proprietatea, p devenind FALSE.

Dacă s-a ieșit pe poarta „i<=n”, înseamnă că au fost „vizitate” toate elementele și nu s-a găsit nici un element care să nu respecte proprietatea, p rămânând astfel TRUE.

Cea mai lungă secvență crescătoare

Problema2. Să se determine lungimea maximă a secvențelor de înălțimi crescătoare și să se afișeze toate secvențele crescătoare de lungime maximă.

Exemplu: Pentru $n=10$ și înălțimile 150, 152, 145, 148, 150, 155, 150, 160, 161, 163 se va afișa:

Lungimea maximă a unei secvențe crescătoare = 4
145, 148, 150, 155 și 150, 160, 161, 163

1. Implementarea algoritmului în limbajul de programare Pascal

```
program demo_secventa_maxima;
const dim=300;
var n,i,l,lmax,j:byte;
    x:array[1..dim] of byte;
begin
    write('n=');readln(n);
    for i:=1 to n do readln(x[i]);
        x[n+1]:=0; {Bordarea vectorului cu o valoare care va întrerupe
                    secvența crescătoare care îl conține pe x[n]}
    lmax:=1;l:=1; {l = lungimea secvenței crescătoare curente}
    for i:=1 to n do
        begin
            if x[i]<=x[i+1] then inc(l)
            else
                begin
                    if l>lmax then
                        lmax:=l;
                    l:=1;
                end;
        end;
    writeln('Lungimea maxima a unei secvente crescatoare=',lmax);
    {Se reparcurge vectorul x și se afișează toate secvențele crescătoare de lungime lmax}
    for i:=1 to n do
        begin
            if x[i]<=x[i+1] then inc(l)
            else
                begin
                    if l=lmax then
                        begin
                            for j:=i-l+1 to i do
                                write(x[j],' ');
                            writeln;
                        end;
                    l:=1;
                end;
        end;
    end.
```

Capcanele lui ZERO

Problema 3. Se consideră un tablou unidimensional (vector) cu n componente ($1 \leq n \leq 50$) numere naturale mai mici sau egale cu 30000. Folosind cele mai mici 3 cifre dintre cifrele unităților componentelor tabloului unidimensional să se genereze cel mai mic număr de trei cifre. Dacă problema nu are soluție se va afișa mesajul 'Nu exista solutie!'.

Exemplu: $n=5$ și $v[1]=3081$; $v[2]=45$; $v[3]=500$; $v[4]=3456$; $v[5]=54$
se va afișa: 104

Timp de execuție: 1 secundă/test.

(InfoKids 2002, Problema 2. „Cel mai mic număr de trei cifre”, clasa a V-a, 100 puncte)

1. Implementarea algoritmului în limbajul de programare Pascal

```
program INFOKIDS_2002;
const dim=50;
type vector=array[1..dim] of integer;
var n,i,z:byte;sortat:boolean;aux:integer;
    x:vector;      {În vectorul x se memorează numerele tastate}
    u:vector;      {În vectorul u se memorează cea mai puțin semnificativă cifră a
                    fiecărui element din x. u[i] conține cifra unităților a numărului
                    memorat în elementul x[i]}

begin
  write('n=');readln(n);
  for i:=1 to n do
    begin
      write('x[' ,i,']=');readln(x[i]);
      u[i]:=x[i] mod 10; {u[i] ← cifra unităților a elementului x[i]}
    end;
    {Se sortează crescător vectorul u, utilizând metoda Bubble Sort, astfel încât cele
    mai mici cifre să apară la începutul vectorului u}
  repeat
    sortat:=true;    for i:=1 to n-1 do
      if u[i]>u[i+1] then
        begin
          aux:=u[i];
          u[i]:=u[i+1];
          u[i+1]:=aux;
          sortat:=false;
        end;
  until sortat;
    {Evitarea capacanelor lui ZERO, ținând cont de faptul că 000 sau 001
    sau 012 nu sunt numere formate din 3 cifre}
  z:=0; {În variabila z se calculează numărul de zerouri din primele trei elemente ale vec-
        torului sortat u}
  for i:=1 to 3 do if u[i]=0 then inc(z);
  case z of
    0: for i:=1 to 3 do write(u[i]);
    3: writeln('Nu exista solutie!');
    1: write(u[2],u[1],u[3]);
    2: write(u[3],u[1],u[2]);
  end;
end.
```


2. Testarea programului
Testarea se face conform datelor din tabelul 54.

Tabelul 54

Nr. test	n	Date de intrare - Vectorul x	Cel mai mic număr de 3 cifre
1.	4	2057, 20, 570, 102	200
2.	5	112, 5, 708, 308, 8010	205
3.	6	75, 250, 78, 800, 45, 30	Nu există soluție
4.	7	1002, 2001, 5071, 245, 784, 2002, 53	112
5.	8	25, 702, 805, 8002, 409, 205, 25, 78	225

Codificarea informației

Problema 4. Conversia din baza 10 în baza 2

Să se convertească numărul natural n , $n \leq 2\,000\,000\,000$, din baza 10 în baza 2.

1. Analiza problemei

Cifrele numărului binar se obțin prin împărțiri succesive la 2 ale numărului zecimal. Algoritmul a fost descris în lecția 4 din partea a II-a. Cifrele binare se vor memora, pe măsura extragerii lor, într-un vector (b2). Deoarece prima cifră extrasă este cea mai puțin semnificativă, pentru a obține reprezentarea binară corectă va trebui „să răsturnăm” vectorul cu cifrele binare.

2. Implementarea algoritmului în limbajul de programare Pascal

```

program baza10_baza2;
const dim=100;
var n:longint; i,u:integer; c:byte;
    b2:array[1..dim] of byte; {Cifrele binare ale numărului n convertit în baza 2}
begin
    write('Tastati nr in baza 10:'); readln(n);
    u:=0; {Indicele ultimului element completat în vectorul b2}
    while (n<>0) do {Se extrag cifrele binare prin împărțiri succesive la 2}
    begin
        c:=n mod 2;
        inc(u); b2[u]:=c;
        n:=n div 2;
    end;

```

{ Pentru a „răsturna” vectorul b2 se inversează fiecare element b2[i] cu elementul simetric b2[u-i+1] utilizând variabila auxiliară c (metoda „paharelor”). Este ca și cum am plasa o oglindă în mijlocul numărului. }

```

for i:=1 to u div 2 do
begin
    c:=b2[i]; b2[i]:=b2[u-i+1]; b2[u-i+1]:=c;
end;
for i:=1 to u do write(b2[i]);
end.

```

Exemplu: $2\,000\,000\,000_{(10)} = 1110111001101011001010000000000_{(2)}$

Problema 5. Codul GRAY

Codul Gray se construiește în felul următor:

- Se pleacă de la o listă care conține un 0 și un 1.
- La fiecare pas se oglindește lista, apoi se adaugă în fața primei jumătăți prefixul 0 și în fața celei de-a doua jumătăți prefixul 1.

Exemplu:

Se pleacă cu următoarea listă:

1. 0

2. 1

Apoi se oglindește lista anterioară obținând:

1

0

Se adaugă prefixele:

1. 00

2. 01

3. 11

4. 10

Continuând procedeul obținem:

1. 000 (Codul Gray de rang 1)

2. 001

3. 011

4. 010

5. 110

6. 111

7. 101

8. 100 (Codul Gray de rang 8)

Fiecare număr din codul GRAY poate fi privit ca reprezentarea în baza 2 a unui număr natural. Se cere să se găsească numărul zecimal a cărui scriere în baza 2 este al n-lea număr din codul GRAY, cu n dat de la tastatură. Numărul cerut se va afișa pe ecran.

Exemple:

1. $n = 8 \Rightarrow 4$; 2. $n = 17 \Rightarrow 24$ 3. $n = 36 \Rightarrow 50$

(Marele Premiu al Palatului Național al Copiilor 2003⁴
clasele VII–VIII)

1. Analiza problemei

Codul Gray este un cod binar foarte important, fiind folosit pentru viteza și corectitudinea sa. Să reprezentăm numerele zecimale 0 .. 7 în baza 2 și în codul Gray (tabelul 55).

⁴ **Marele Premiu al Palatului Național al Copiilor** este un Concurs Național de Informatică pentru clasele gimnaziale și de liceu la care participă echipe ale Palatelor și Cluburilor Copiilor din întreaga țară. Concursul este organizat de Palatul Național al Copiilor din București și se află la a XIII-a ediție.

Valoarea zecimală	Cod binar (baza 2)	Cod Gray	Conversie BINAR → GRAY
0	000	000	
1	001	001	
2	010	011	
3	011	010	$0 (0+1)_{\text{mod } 2} (1+1)_{\text{mod } 2} = 010; (1+1=2; 2 \text{ mod } 2=0)$
4	100	110	
5	101	111	
6	110	101	
7	111	100	

Ce putem observa din acest tabel?

1. Două coduri Gray succesive diferă printr-un singur bit; 001 → 011.
2. Codul binar și codul Gray corespunzător au cifrele cele mai semnificative identice.
3. Codul binar și codul Gray corespunzător au același număr de cifre.
4. Cifra de rang i a codului Gray (exclusiv cifra cea mai semnificativă) este egală cu restul împărțirii la 2 a sumei cifrelor de rang $(i-1)$ și i ale codului binar corespunzător: $g_i = (b_{i-1} + b_i) \text{ mod } 2$

Codul binar și codul Gray se vor memora în doi vectori: **b** și **g**.

2. Implementarea algoritmului în limbajul de programare Pascal

Procesul de obținere a codului Gray al unui număr zecimal dat, n , pornind de la reprezentarea în baza 2 a numărului n se implementează astfel:

```
g[1]:= b[1];
```

```
for i:=2 to nc do
```

```
  g[i]:= (b[i-1]+b[i]) mod 2;
```

unde nc reprezintă numărul de cifre al reprezentării binare a numărului zecimal.

```
program cod_GRAY;
```

```
var b,g:array[1..100] of 0..1;
```

```
  c,aux:byte; i,nc:integer;  n,nb,nr:longint;
```

```
begin
```

```
  write('n=');readln(n);
```

```
  if n=0 then write(0)
```

```
  else
```

```
    BEGIN
```

```
      n:=n-1;nc:=0; {Obținerea codului binar}
```

```
      while (n<>0) do
```

```
        begin
```

```
          c:=n mod 2;inc(nc);
```

```
          b[nc]:=c; n:=n div 2;
```



```

end;
for i:=1 to nc div 2 do {Răsturnarea codului binar}
begin
    aux:=b[i];
    b[i]:=b[nc-i+1];
    b[nc-i+1]:=aux;
end;
(* Conversie cod binar -> cod Gray *)
g[1]:=b[1];
for i:=2 to nc do
    g[i]:=(b[i-1]+b[i]) mod 2;
{Obținerea numărului zecimal a cărui reprezentare în baza 2 coincide cu reprezentarea
GRAY din vectorul g}

nr:=0;
for i:=1 to nc do
    nr:=nr*2+g[i];
    write(nr);
END;
end.

```

3. Testarea programului

Testele pentru 100 de puncte

Tabelul 56

Nr. test	Intrare/Ieșire	Nr. test	Intrare/Ieșire
1.	6 $\Rightarrow$ 7	6.	1734 $\Rightarrow$ 1447
2.	7 $\Rightarrow$ 24	7.	732 $\Rightarrow$ 7506
3.	57 $\Rightarrow$ 36	8.	12753 $\Rightarrow$ 10552
4.	129 $\Rightarrow$ 192	9.	32745 $\Rightarrow$ 16412
5.	435 $\Rightarrow$ 363	10.	59873 $\Rightarrow$ 40208

16. Numere uriașe

Povestea șahului

Se spune că străvechiul joc de șah a fost inventat de un înțelept indian, Sessa ibn Dahar, pentru distracția împăratului perșilor, Seram. Împăratul perșilor a fost foarte încântat de joc și dorind să-l răsplătească „împărătește” pe înțeleptul indian, l-a întrebat pe acesta ce ar dori. La insistențele împăratului, înțeleptul a cerut un bob de grâu pentru cea dintâi căsuță a tablei de șah, două boabe de grâu pentru a doua căsuță, patru boabe pentru a treia, opt pentru a patra, ș.a.m.d. până la a 64-a căsuță a tablei de șah.

Împăratul s-a mirat de modestia înțeleptului și și-a chemat vistiernicii să calculeze numărul de boabe pe care trebuie să le dăruiască înțeleptului. Împăratul Seram a fost foarte surprins de răspunsul vistiernicilor.

Pentru a afla de ce a fost atât de surprins împăratul scrieți un program care să calculeze numărul de boabe de grâu datorat de împărat.

1. Analiza problemei

Numărul total de boabe $S = 1 + 2 + 2^2 + 2^3 + \dots + 2^{63} = 18446744073709551615$.

Acum înțelegeți surpriza împăratului! Numărul depășea cu mult rezervele împărăției!

S este un număr cu 20 de cifre care nu poate fi reprezentat utilizând longint, dar nici extended. Pentru a putea reprezenta cu precizie numere „uriaeșe” le vom reprezenta cu ajutorul tablourilor unidimensionale, fiecare cifră ocupând o locație a tabloului.

- Dimensiunea tabloului trebuie să fie suficient de mare ca să poată „încăpea” cel mai mare număr posibil pentru problema care se rezolvă.
- Tabloul se inițializează cu 0.
- Primul element va conține numărul efectiv de cifre al numărului uriaș.
- Începând cu al doilea element se memorează cifrele numărului răsturnat; în al doilea element al vectorului se va memora cifra unităților.

Operațiile cu numere „uriaeșe” se desfășoară după aceleași reguli ca numerele obișnuite. De exemplu, la adunare, se adună cifrele de același rang și se memorează restul împărțirii la 10 a sumei cifrelor de același rang, iar dacă există transport, acesta se adună la suma cifrelor rangului următor.

Adunarea a două numere
$\begin{array}{r} 1 \leftarrow \text{transport de la} \\ \text{rangul unităților} \\ 124+ \\ \underline{69} \\ 193 \end{array}$

	Nr. cifrelor a[0]	Cifra unităților a[1]	Cifra zecilor a[2]	Cifra sutelor a[3]	Cifra miilor	...
a	3	4	2	1	0	0
b	2	9	6	0	0	0
c	3	3	9	1	0	0

transport

2. Implementarea algoritmului în limbajul de programare Pascal

```

program istoria_sahului;
type numere_urias=array[0..1000] of integer;
var s,ti:numere_urias; i,j,transport,c,lim:integer;
begin
  fillchar(s,sizeof(s),0); fillchar(ti,sizeof(ti),0);
  ti[0]:=1; ti[1]:=1; {Termenul general (Ti) al sumei se inițializează pe 1}
  s[0]:=1;s[1]:=1;    {Suma S se inițializează pe 1}
  for i:=2 to 64 do

```



```

BEGIN
    { Calculul termenului de rang i cu ajutorul formulei de recurență }
     $t_i = t_{i-1} * 2$ 
    transport:=0;
    for j:=1 to ti[0] do
        begin
            ti[j]:=ti[j]*2+transport; transport:=ti[j] div 10;
            ti[j]:=ti[j] mod 10;
        end;
    if transport=1 then { Crește numărul de cifre cu 1 }
    begin inc(ti[0]); ti[ti[0]]:=transport; end;
    { Calculul sumei parțiale s = s + ti }
    transport:=0;
    if s[0]>ti[0] then lim:=s[0] else lim:=ti[0];
    for j:=1 to lim do
        begin
            c:=s[j]+ti[j]+transport; s[j]:=c mod 10;
            transport:=c div 10;
        end;
    if transport=1 then
    begin inc(s[0]); s[s[0]]:=transport end;
    END;
    for i:=s[0] downto 1 do write(s[i]);
end.

```

Produsul „uriaș” al două numere de tip longint

```

type huge_numbers=array[0..1000] of integer;
var a,b,c: huge_numbers;
    cifra,na,nb,i,t,j: longint;
begin
    write('na='); readln(na);
    i:=0; { Extragerea cifrelor nr. na în vectorul a }
    while (na<>0) do
        begin
            cifra:=na mod 10; inc(i);
            a[i]:=cifra; na:=na div 10;
        end;
    a[0]:=i;
    write('nb='); readln(nb);
    i:=0;
    while (nb<>0) do begin
        cifra:=nb mod 10; inc(i);
        b[i]:=cifra; nb:=nb div 10; end;
    b[0]:=i;

```



```

{Obținerea produsului  $a * b$  în vectorul c}
fillchar(c, sizeof(c), 0);
c[0] := a[0] + b[0] - 1;
for i := 1 to a[0] do
  for j := 1 to b[0] do
    c[i+j-1] := c[i+j-1] + a[i] * b[j];
{Deoarece produsele parțiale obținute în vectorul c pot fi formate din mai multe cifre, în
secvența următoare se determină și se propagă transportul către rangurile superioare astfel încât
toate elementele vectorului c să fie formate dintr-o singură cifră.}
t := 0;
for i := 1 to c[0] do
  begin
    c[i] := c[i] + t;
    t := c[i] div 10;
    c[i] := c[i] mod 10;
  end;
if t <> 0 then begin inc(c[0]); c[c[0]] := t; end;
for i := c[0] downto 1 do write(c[i]);
end.

```

Secvența de calcul a diferenței dintre două numere „uriaeșe” $a = a - b$; $a > b$

```

impr := 0;
for i := 1 to a[0] do
  begin
    a[i] := a[i] - b[i] - impr;
    if a[i] < 0 then
      begin
        impr := 1; a[i] := a[i] + 10;
      end
    else impr := 0;
  end;
{Decrementarea numărului de cifre al diferenței până când cea mai semnificativă cifră
a diferenței este <> 0}
while a[a[0]] = 0 do dec(a[0]);

```


Probleme propuse

1. Sistemul de numerație Fibonacci (*Orice număr natural poate fi scris ca o sumă de termeni Fibonacci.*)

Să se scrie numărul natural $n \leq 2\,000\,000\,000$ ca o sumă formată dintr-un număr minim de termeni Fibonacci.

Indicații: Se generează într-un vector toate numerele Fibonacci, mai mici sau egale cu n . Se caută cel mai apropiat număr Fibonacci de n , acesta fiind primul termen al descompunerii lui n în număr minim de termeni Fibonacci. Procedeu se reia pentru diferența dintre n și termenul Fibonacci găsit, ș.a.m.d.

2. Numere „norocoase”

Se consideră șirul numerelor impare: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, ... Primul număr impar diferit de 1 din șir fiind 3, se elimină tot al treilea număr din șir. După această procedură rămân numerele: 1, 3, 7, 9, 13, 15, 19, 21, 25, 27, 31, ... Primul număr din șir mai mare decât 3 fiind 7, se elimină tot al șaptelea număr din șir, rămânând: 1, 3, 7, 9, 13, 15, 21, 25, 31, ... ș.a.m.d.

Să se scrie în fișierul text „lucky.out” numerele „norocoase”, mai mici sau egale cu n , după reluarea procedurii de mai sus de k ori. n și k se citesc din fișierul text „lucky.in”, pe o singură linie, separate printr-un spațiu.

Restricții: n și k sunt numere naturale, $n \leq 60\,000$, $k < n$.

3. Bile

Pe o masă de tip biliard sunt dispuse n bile numerotate de la 1 la n ($n \leq 30$ citit de la tastatură). Bilele sunt eliminate de pe masă una câte una, fiecare bilă eliminată fiind luată de un „băiat de bile” și pusă într-un șir. Băiatul de bile încearcă să plaseze fiecare bilă nou eliminată exact la mijlocul șirului de bile deja format (numărul de bile din față să fie egal cu numărul de bile de după aceasta). Când acest lucru nu este posibil, atunci el o plasează la sfârșitul șirului de bile.

Scrieți un program care să citească de la tastatură numărul de bile, apoi ordinea finală a bilelor în șir și care să afișeze pe ecran numerele bilelor, în ordinea în care au fost eliminate de pe masa de biliard, separate prin spații.

Exemplu: Pentru $n = 7$ și șirul final = 1 7 2 5 3 4 6

Se va afișa: Bilele sunt eliminate în ordinea 1 3 7 4 2 6 5.

Timp maxim de execuție: 1 secundă/test.

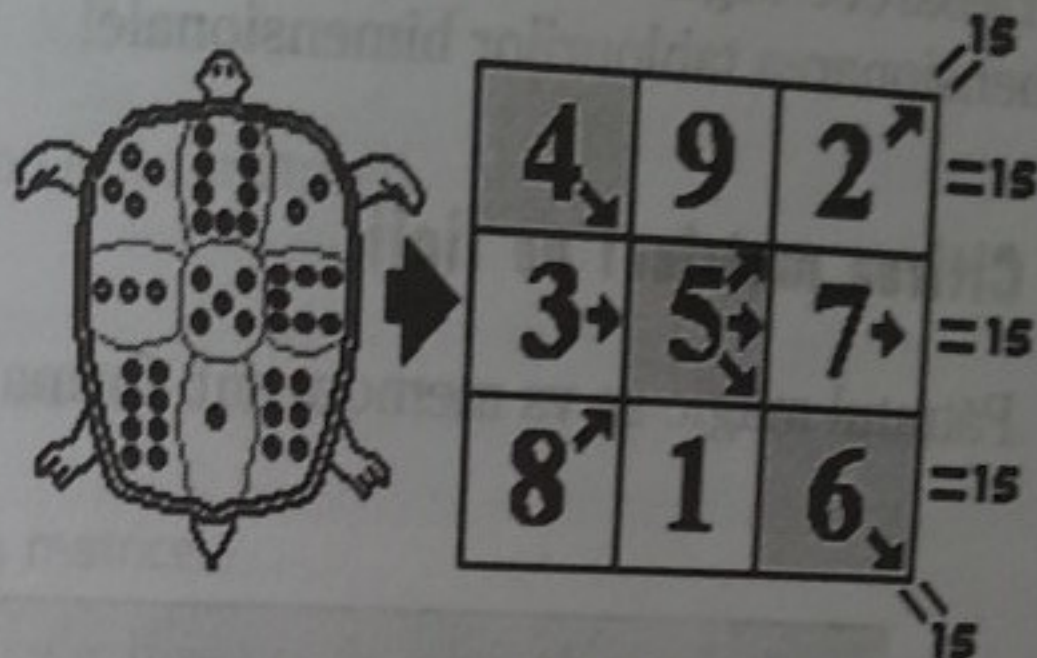
(Concursul național de soft „Grigore Moisil”⁵, clasele V–VI)

⁵ **Concursul Național de Soft „Grigore Moisil”** este organizat de Palatul Copiilor din Lugoj și se află la a X-a ediție. Concursul se desfășoară pe două secțiuni: programare (clasele V–VI, VII–VIII, IX–X, XI–XII) și comunicări științifice (soft de aplicații și jocuri, soft educațional și pagini Web).

Tablouri bidimensionale (matrice)

Legenda pătratului magic

Conform legendelor chinezești, acum 4000 de ani, o broască țestoasă uriașă a ieșit din apele fluviului Lo din centrul Chinei. Pe carapacea broaștei se afla un model format din 3 linii și 3 coloane, la intersecția cărora era desenat din puncte câte un număr.



Pe orice direcție s-au adunat numerele s-a obținut aceeași sumă 15 : 15 zile în fiecare din cele 24 cicluri ale anului solar chinezesc. Pătratul magic reprezintă baza teoretică a școlii Feng Shui.

Pătratul de pe carapacea broaștei țestoase este primul pătrat magic cunoscut, un tablou bidimensional de dimensiune 3×3 , 3 linii și 3 coloane.

Un tablou bidimensional de dimensiune $n \times m$ este o succesiune de n vectori de dimensiune m .

	1	2	...	m
1	a[1, 1]	a[1, 2]	...	a[1, m]
2	a[2, 1]	a[2, 2]	...	a[2, m]
...	...	...	...	...
n	a[n, 1]	a[n, 2]	...	a[n, m]

- Toate elementele unui tablou bidimensional sunt de același tip.
- Se poate accesa orice element al tabloului utilizând notația $a[i, j]$, unde i este indicele liniei, iar j este indicele coloanei.
- Tabloul bidimensional cu $n = m$ se numește *pătratic*.

Declararea unei matrice

const

max_linii=10;

max_coloane=10;

type

matrice=array [1..max_linii, 1..max_coloane] of integer;

var

a:matrice;

Ce spațiu va ocupa variabila *a* în memorie?
 $\text{max\_linii} * \text{max\_coloane} * \text{sizeof}(\text{tip\_bază}) = 10 * 10 * 2 = 200 \text{ Bytes}$
 Deoarece segmentul de date este limitat la 65536 Bytes, trebuie multă atenție la dimensionarea tablourilor bidimensionale!

Citirea matricei pe linii

Pătratul magic se va memora sub forma șirului: 4, 9, 2, 3, 5, 7, 8, 1, 6.

Tabelul 59

<i>i</i> = indicele de linie	<i>j</i> = indicele de coloană	Elementul <i>a</i> [<i>i</i> , <i>j</i>]
1	1	<i>a</i> [1, 1] = 4
	2	<i>a</i> [1, 2] = 9
	3	<i>a</i> [1, 3] = 2
2	1	<i>a</i> [2, 1] = 3
	2	<i>a</i> [2, 2] = 5
	3	<i>a</i> [2, 3] = 7
3	1	<i>a</i> [3, 1] = 8
	2	<i>a</i> [3, 2] = 1
	3	<i>a</i> [3, 3] = 6

Citirea matricei se va face utilizând două cicluri FOR, ciclul din interior având variabila de control care se schimbă cel mai des, adică indicele de coloană *j*.

```
for i:=1 to n do
  for j:=1 to m do
    readln(a[i, j]);
```

Citirea matricei pe coloane

Pătratul magic se va memora sub forma următorului șir: 4, 3, 8, 9, 5, 1, 2, 7, 6. Indicele liniei se va schimba mai des decât indicele coloanei.

```
for j:=1 to m do
  for i:=1 to n do
    readln(a[i, j]);
```

Problema 1. Pătratul magic

Se citește un tablou bidimensional pătratic, $n \times n$ ($n \leq 50$). Să se verifice dacă este un pătrat magic.

1. Implementarea algoritmului în limbajul de programare Pascal

```
program patrat_magic;
const dim = 50;
```



```

var a : array [1..dim,1..dim] of integer;
    n,i,j: byte;
    s_magic,s:longint; MAGIC:boolean;

begin
    write('n='); readln(n);
    for i:=1 to n do {Citirea pe linii a matricei}
        for j:=1 to n do
            begin
                write('a[' ,i ,', ' ,j ,']='); readln(a[i,j]);
            end;
        for i:=1 to n do {Afişarea matricei}
            begin
                for j:=1 to n do
                    write(a[i,j]:6);
                writeln;
            end;
        MAGIC:=true;
        s_magic:=0; {Calculul sumei elementelor de pe diagonala principală}
        for i:=1 to n do
            s_magic:=s_magic+a[i,i];
        s:=0;
        for i:=1 to n do {Calculul sumei elementelor de pe diagonala secundară}
            s:=s+a[i,n-i+1];
        if s<>s_magic then MAGIC:=false
        else
            BEGIN
                i:=1; {Calculul sumei elementelor pe linii}
                while MAGIC and (i<=n) do
                    begin
                        s:=0;
                        for j:=1 to n do
                            s:=s+a[i,j];
                        if s<>s_magic then MAGIC:=false;
                        inc(i);
                    end;
                j:=1; {Calculul sumei elementelor pe coloane}
                while MAGIC and (j<=n) do
                    begin
                        s:=0;
                        for i:=1 to n do
                            s:=s+a[i,j];
                        if s<>s_magic then MAGIC:=false;
                        inc(j);
                    end;
            END;
        if MAGIC then write('Este patrat magic cu S=',s_magic)
        else write('Nu este patrat magic'); end.

```


Observație. Un algoritm eficient trebuie să asigure ieșirea din bucla de verificare a liniilor sau coloanelor în momentul în care o linie sau o coloană nu respectă proprietatea. Suma elementelor de pe diagonala principală (*s_magic*) a fost luată ca reper. Variabila logică *MAGIC* va fi poziționată pe *FALSE* la prima sumă diferită de *s_magic*, întrerupând verificarea celorlalte direcții.

Problema 2. Cel mai văzut concurent

La concursul *InfoKids* – secțiunea programare (clasa a VIII-a) s-au prezentat *n* elevi ($1 \leq n \leq 50$), codificați prin 1, 2, ..., *n*. Cu această ocazie s-au întâlnit vechi cunoștințe (de la alte concursuri), întocmindu-se un tabel în care fiecare elev a indicat pe care din cei *n* elevi i-a mai „văzut” cu alte prilejuri. Să se precizeze care este elevul cel mai „văzut”.

Date de intrare: În fișierul text cu numele *KIDS8.IN* pe prima linie se află *n*, iar pe următoarele *n* linii se află elevii „văzuți” de fiecare dintre concurenți (în ordinea 1, 2, ..., *n*).

Date de ieșire: În fișierul text cu numele *KIDS8.OUT* se va scrie pe prima linie codul elevului cel mai văzut (dacă există mai mulți elevi se va scrie codul lor pe același rând cu câte un spațiu între ele).

Exemplu:

KIDS8.IN

KIDS8.OUT

5

1 4

1 2 3

2 1 5 4

1 3 4

4

1 5 4

Timp de execuție: 1 secundă/test.

(InfoKids 2002 — Concursul Național de Informatică pentru clasele I-IX;
Problema 1 „Văzut” 100 puncte, clasa a VIII-a)

1. Analiza problemei

Ca arhitecți ai propriilor noastre structuri de date putem construi o matrice în care fiecare linie să fie asociată unui elev. Matricea se inițializează pe 0 și coloanele fiecărei linii se completează cu elevii văzuți de elevul asociat liniei.

2. Implementarea algoritmului în limbajul de programare Pascal

```
program vazut;
const dim = 50;
var a : array [1..dim,1..dim] of byte;
    freq : array [1..dim] of byte;
    n , i, j, max : byte;
    fi, fo: text;
begin
    assign (fi, 'kids.in'); reset (fi);
    assign (fo, 'kids8.out'); rewrite (fo);
    fillchar(a, sizeof(a), 0); fillchar(freq, sizeof(freq), 0);
    read (fi, n);
    for i:=1 to n do
```



```

begin
  j:=0;
  repeat
    (Linile fiind de lungime variabilă va trebui să apelăm la funcția seekeoln
    pentru detectarea sfârșitului de linie)
    j:=j+1;
    read(fi,a[i,j]);
  until seekeoln(fi);
end;
for i:=1 to n do
  for j:=1 to n do
    if a[i,j]<>0 then
      freq[a[i,j]]:=freq[a[i,j]]+1;
max:=0;
for i:=1 to n do
  if freq[i]>max then
    max:=freq[i];
for i:=1 to n do
  if freq[i]= max then
    write(fo,i,' ');
close(fi);close(fo);end.

```

18. Șiruri de caractere (string)

Programele nu prelucrează doar informație numerică ci și șiruri de caractere.

Declaraarea și inițializarea unei variabile de tip string

```
var s:string;
```

```
...
```

```
s:='Test';
```

Variabila s va ocupa o zonă de memorie formată din 256 Bytes, din care se vor utiliza doar 5 Bytes.

Tabelul 58

Lungime șir	'T'	'e'	's'	't'	256-5=251 Bytes neutilizați
4	84	101	115	116	
s[0]	s[1]	s[2]	s[3]	s[4]	

Lungimea efectivă a unui șir de caractere se obține apelând funcția **length**.
s[i] conține codul ASCII al caracterului de rang i în șirul s.

Exemple:

1. Programul următor numără câte vocale și câte consoane conține un text introdus de la tastatură și format din maxim 80 de caractere.


```

program demo_siruri;
CONST
  LITERE:set of char=['A'..'Z','a'..'z'];
  VOCALE:set of char= ['A','E','I','O','U','a','e','i',
    'o','u'];
VAR
  CONSOANE:set of char;s:string; n,i,v,c:integer;
begin
  CONSOANE:=LITERE-VOCALE;
  write('Tastati sirul (max 80 caractere):');readln(s);
  if s='' then begin write('Sir vid'); readln; exit end;
  n:=length(s);
  if n>80 then begin write('Sir prea lung'); readln; exit end;
  v:=0;c:=0;
  for i:=1 to n do
    if s[i] in VOCALE then inc(v)
    else
      if s[i] in CONSOANE then inc(c);
  write('Textul are ', v, ' vocale si ', c, ' consoane');
end.

```

2. Programul al doilea afișează frecvența literelor dintr-un șir de maxim 255 de caractere, introdus de la tastatură.

```

program demo_siruri;
VAR
  s:string;n,i:integer; ch:char; freq:array['A'..'Z'] of byte;
begin
  write('Tastati sirul:');readln(s);
  for ch:='A' to 'Z' do {Inițializarea vectorului cu frecvența literelor A..Z}
    freq[ch]:=0;
  for i:=1 to n do
    for ch:='A' to 'Z' do
      if upcase(s[i])=ch then inc(freq[ch]);
  writeln('Frecvențele literelor in text');
  for ch:='A' to 'Z' do
    write(ch,'=',freq[ch],', ',');
end.

```

19. Tipul înregistrare (record)

Tablourile sunt structuri de date omogene, toate elementele fiind de același tip. Dacă dorim să îmbinăm într-o structură date de tipuri diferite putem utiliza **tipul înregistrare**.


```

type
  nume_tip=RECORD
      câmp1:tip1;
      câmp2:tip2;
      ...
      câmpn:tipn;
  END;

```

Utilizând tipul înregistrare putem memora într-o singură variabilă (e) toate informațiile necesare despre un elev. Fiecare câmp poate fi accesat individual utilizând sintaxa: *nume_variabilă.nume_câmp*.

Putem obține o „scurtătură” dacă utilizăm instrucțiunea **with**.

```

program elev;

```

```

type
  info_elev=RECORD
      nume:string[40];
      varsta:byte;
      inaltime:byte;
  END;
var
  e:info_elev;
begin
  write('Nume:');readln(e.nume);
  write('Varsta:');readln(e.varsta);
  write('Inaltimea:');readln(e.inaltime);
  with e do
    writeln('Elevul ',nume,' in varsta de ',varsta,' ani are
      inaltimea de ',inaltime,' cm'); end.

```

20. Arhitecturi complexe de date

Putem construi structuri complexe de date combinând tipul RECORD cu tipul ARRAY.

	nume	vârsta	înălțime
Elevul 1	Popescu Bogdan	13	165
Elevul 2	Stoica Dragoș	13	169
...	...	...	...
Elevul n	Ionescu Ioana	13	163

```

program structuri_complexe;
const dim=300;
type

```



```

info_elev=RECORD
                                nume:string[40];
                                varsta:byte;
                                inaltime:byte;
                                END;
var   e:array[1..dim] of info_elev;
      n,i:integer;s,h_med:real;
begin
  write('n=');readln(n);
  for i:=1 to n do
  BEGIN
    write('Nume:');readln(e[i].nume);
    write('Varsta:');readln(e[i].varsta);
    write('Inaltimea:');readln(e[i].inaltime);
  END;
  s:=0;
  for i:=1 to n do
    s:=s+e[i].inaltime;
  if n<>0 then
    begin
      h_med:=s/n;
      writeln('Inaltimea medie=',h_med:6:2);
    end
  else writeln('Nici un elev!');
end.

```

Cu tipul înregistrare am ajuns la poarta PROGRAMĂRII ORIENTATE PE OBIECTE, tipul *obiect* declarându-se ca un tip înregistrare care îmbină datele cu metodele (subprogramele) care prelucrează datele.

ÎN LOC DE RECAPITULARE

DACĂ AȚI SCRIS UN PROGRAM

- ✓ Corect, care trece toate testele cu date de intrare caracteristice, extreme și ilegale
- ✓ Logic și clar
- ✓ Structurat
- ✓ Bine documentat
- ✓ Ușor de înțeles, de depanat și de modificat
- ✓ Optimizat din punct de vedere al timpului de execuție și al spațiului de memorie utilizat,

... ATUNCI VEȚI DEVENI BUNI PROGRAMATORI!

ANEXĂ

Elementele de bază ale limbajului Pascal

I. Tipuri de date

Tipuri de date

Tabelul 59

Tipul de date	Definire	Domeniul de valori	Operatori	Observații
Tipul întreg	Byte	0 ... 255	+, - *, /, Div, Mod +, <, >, <=, >=, <>, =	constante Maxint = 32767 Maxlongint = 2147483647
	Word	0 ... 65535		
	Shortint	-128 ... 127		
	Integer	-32768 ... 32767		
	Longint	-2147483648... 2147483647		
Tipul logic	Boolean	False, True	Not, And, Or <, >, <=, >=, <>, =	
Tipul caracter	Char	orice caracter ASCII	<, >, <=, >=, <>, =	
Tipul real	Real	2.9E-39 ... 1.7E38	+, - *, /, <, >, <=, >=, <>, =	11-12 cifre semnificative 7-8 15-16 19-20
	Single	1.5E-45 ... 3.4E38		
	Double	5.0E-324 ... 1.7E308		
	Extended	3.4E-4932...1.1E4932		
Tipul șir de caractere	String [L]	cel mult L caractere ASCII	+ (concatenare) <, >, <=, >=, <>, =	L - numărul de caractere din șir, implicit 255
Tipul tablou	Array [S1] of tipb	cel mult numărul de elemente din S1 de tipul tipb	toți operatorii acceptați de tipul de bază tipb	Tablou de tip vector
	Array [S1, S2] of tipb	cel mult S1 × S2 elemente de tipul tipb		Tablou de tip matrice
Tipul articol	Record câmpuri end;	fiecare câmp ia valori în domeniul tipului de date asociat	toți operatorii acceptați de tipul câmpului	Operațiile se execută la nivel de câmp
Tipul mulțime	Set of tipb	cel mult 256 de elemente	* intersecție + reuniune - diferență IN apartenență <, >, <=, >=, <>, =	Tipul de bază, tipb este tip ordinal

II. Instrucțiuni structurate

Tipuri de instrucțiuni structurate

Tabelul 60

Instrucțiune	Sintaxa	Observații
Atribuire	variabilă: = expresie;	
Bloc de instrucțiuni	Begin End	
Decizie	If condiție Then instrucțiune sau Bloc Else instrucțiune sau Bloc;	– ramura Else poate să lipsească; – separatorul punct virgulă (;) plasat după instrucțiunea/blocul din Then închide structura
Selecție	Case selector Of C1: instrucțiune/bloc; C2: instrucțiune/ bloc; Else instrucțiune End;	– selectorul este de tip ordinal – C1, C2 ... sunt etichete de același tip cu selectorul – ramura Else poate lipsi
Repetiție cu condiția la sfârșit	Repeat instrucțiuni until condiție;	– instrucțiunile se execută cel puțin o dată
Repetiție cu condiția la început	While condiție Do instrucțiune/bloc;	
Repetiție cu număr cunoscut de pași	For contor: = vi To/Downto vf Do instrucțiune/bloc;	– vi: valoarea inițială a contorului – vf: valoarea finală a contorului – pasul este 1 pentru contorul crescător (To) și -1 pentru contorul descrescător (Downto)

III. Funcții și proceduri de sistem

Limbajul de programare pune la dispoziția programatorului o bibliotecă de sistem (unitul System) din care se pot apela subprograme care rezolvă operații cu numere sau șiruri de caractere. Există două tipuri de subprograme:

- subprograme de *tip funcție* — acestea se apelează prin numele lor și lista de parametri; în expresii sau instrucțiuni de atribuire; o funcție are un tip asociat;
- subprograme de *tip procedură* — acestea se apelează prin numele lor și lista de parametri direct în blocul de instrucțiuni.

Funcții și proceduri

Tabelul 61

Denumire	Sintaxa	Tip F - funcție P - procedură	Semnificație
ABS	Abs (x)	F	calculează valoarea absolută (modulul) lui x întreg sau real
COS	Cos (x)	F – real	funcție trigonometrică, x real
DEC	Dec (x)	P	decrementează (scade cu o unitate) valoarea lui x întreg
FRAC	Frac (x)	F – real	calculează partea zecimală a lui x real
INC	Inc (x)	P	incrementează (crește cu o unitate) valoarea lui x întreg
INT	int (x)	F – real	calculează partea întreagă a lui x real
LN	Ln (x)	F – real	calculează logaritm natural din x real
ODD	Odd (x)	F – logic	dacă x este impar întoarce valoarea logică true
RANDOMIZE	Randomize	P	pornește algoritmul de generare numere aleatoare
RANDOM	Random (N)	F – real	generează valori aleatoare între 0 și N
ROUND	Round (x)	F – întreg	calculează cel mai apropiat întreg față de x real
SIN	Sin (x)	F – real	funcția trigonometrică, x real
SQR	Sqr (x)	F	calculează valoarea lui x ² de tip întreg sau real
SQRT	Sqrt (x)	F – real	calculează radicalul lui x real (rădăcina pătrată)
TRUNC	Trunc (x)	F – întreg	calculează cel mai mic întreg față de x real

Funcții și proceduri pentru șiruri de caractere

Tabelul 62

Denumire	Sintaxa	Tip F - funcție P - procedură	Semnificația parametrilor	Ce realizează
CONCAT	Concat (S1, S2 ...)	F – string	S1 , S2 subșiruri de caractere	Concatenează șirurile S1 , S2 într-un șir nou
COPY	Copy (S, poz, lg)	F – string	S – șirul de bază poz – poziția în S lg – lungimea subșirului	Copiază din S , începând de la poz , subșirul de lungime lg
DELETE	Delete (S, poz, lg)	P		Șterge din S , începând de la poz , subșirul de lungime lg
INSERT	Insert (S1, S, poz)	P	S1 – subșirul de inserat S – șirul de bază poz – poziția în șirul de bază	Inserază în S , de la poziția poz , subșirul S1
LENGTH	Length (S)	F – întreg	S – șirul de bază	Determină lungimea (numărul de caractere) șirului S
POS	Pos (S1, S)	F: byte	S1 – subșirul S – șirul de bază	Determină poziția la care începe subșirul S1 în șirul S ; dacă S1 nu există în S , valoarea este zero

Tabelul 62 (continuare)

Denumire	Sintaxa	Tip F - funcție P - procedură	Semnificația parametrilor	Ce realizează
STR	Str (N, S)	P	N – variabilă de tip numeric S – variabilă de tip string	Transformă valoarea numerică a variabilei N în valoarea de tip string a variabilei S
VAL	Val (S, N, Cod)	P	S – variabilă de tip string N – variabilă de tip numeric Cod – variabilă de tip întreg care oferă informații despre rezultatul conversiei	Transformă valoarea variabilei de tip string S, în valoarea numerică a variabilei N Cod = valoarea 0 dacă tot stringul este transformat sau Cod = numărul de caractere transformate

Tabela codurilor ASCII

Zeci- mal	Hexaze- cimal	Caracter	Zeci- mal	Hexaze- cimal	Caracter	Zeci- mal	Hexaze- cimal	Caracter	Zeci- mal	Hexaze- cimal	Caracter
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	-	67	43	C	99	63	c
4	4	EOT	36	24	\$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	"	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	"	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	

Bibliografie

- Cormen, Th. H.; Leiserson, Ch. E., Rivest, R. R. — *Introducere în algoritmi*, Editura Agora, 2002
- Cristian, V.; Eugenia Kalisz; Irina Athanasiu; Pănoiu, Al. — *Turbo Pascal*, Editura Teora, 1992
- Dertouzos, M. — *CE VA FI — Cum vom trăi în lumea nouă a informației*, Editura Tehnică, 2000
- Frâncu, C. — *Psihologia concursurilor de informatică*, Editura L&S, 1998
- Gardner, M. — *Amuzamente matematice*, Editura Științifică, București, 1968
- Giumale, C. și colectiv — *Proiectarea și analiza algoritmilor*, Editura ALL, 1996
- Mioara Gheorghe — *Caiet de laborator pentru clasa a IX-a*, Editura ALL, București, 2001
- Horowitz Ellis; Sanni Sartaj; Rajeskar Sanguthevar — *Computes Algorithmis*, Freemau, 1998
- Mariana Kisk; Kisk, D. — *Turbo Pascal pentru copii și nu numai ...*, Editura ICPE, 1997
- Knuth, D. E. — *Arta programării calculatoarelor — Volumul 1, Algoritmi Fundamentali*, Editura Teora, 1999
- Marcus, S. — *Omul și oceanul său comunicațional în secolul 20*, 4-9/2000
- Sorin, T. — *Informatica — Varianta Pascal (manual pentru clasa a IX-a)*, Editura L&S, 2000
- Pratta, S. — *Manual de programare în C++*, Editura Teora, 2001
- O'Brien, S. — *Turbo Pascal Version 6.0 — The Complete Reference*, Borland International Inc.
- Tischer, M. — *La Bible TURBO PASCAL*, Edition Micro Application, Paris, 1990
- Bobancu, V. — *Caleidoscop matematic*, Editura Niculescu, 2001
- *** Colecția gazeta de informatică 1991-1997
- *** Colecția GINFO 1998-2003
- <http://mathforum.org>
- <http://sunflower.singnet.com>

Cuprins

Mai aproape de copii, mai aproape de calculator	3
PARTEA I. MODELAREA GÂNDIRII ALGORITMICE	5
1. Cine muncește mai mult: noi sau calculatorul?	5
2. Și calculatorul învață!	8
3. Memoria – o resursă limitată	10
4. Time out!	11
5. Poveste pe gustul fiecăruia	13
6. Reprezentarea algoritmilor	15
7. Să gândim structurat	18
8. Algoritmi secvențiali. Regula celor trei pahare	21
Descifrarea algoritmilor	24
9. Adevărat sau fals	26
10. Decizii în cascadă	29
11. Din nou la ZooCarnaval	36
12. Selecția	38
13. Pas cu pas prin algoritm	41
14. Repetiția condiționată	45
Repetă până când	45
Repetă cât timp	47
15. O înmulțire ciudată	51
16. O problemă de geometrie... „în așteptare”	53
17. Algoritmul lui Euclid	56
18. Repetiția cu contor	58
19. Trasee turistice	62
20. Organizarea datelor	64
21. Din nou trasee turistice	70
22. Ciurul lui Eratostene	73
23. Viruși în doi vectori	75
24. Viruși într-o matrice	77
25. Rucsacul și algoritmul lacom	80
26. Cum ordonăm datele	82
27. Dacă ești mic, stai în față!	84
28. Numărarea predecesorilor	86
29. O alegere dificilă	88
30. Deviza cuceritorului	92
Teste – răspunsuri	96
În loc de recapitulare	96
PARTEA a II-a. ARTA DE A PROGRAMA	97
1. Limbaje programare	97
Istoria limbajelor de programare	98
Limbajul mașină	98
Limbajul de asamblare (1950)	98
Limbajul de nivel înalt (1970)	98
Limbajul natural vorbit	99

Limbaje de nivel înalt	99
Basic	99
Pascal	100
C	101
C++	101
2. Limbajul de programare PASCAL	101
Vocabularul limbajului Pascal	101
3. Copii celebri din lumea calculatoarelor	102
4. În culisele limbajului de programare	103
Mașina „cu două degete”	103
Reprezentarea internă a informației nenumerice	104
Sisteme de numerație	104
Sisteme de numerație poziționale	104
Conversia dintr-o bază de numerație în alta	105
Conversia unui număr natural din baza 10 în baza 2	105
Conversia unui număr natural din baza 2 în baza 10	107
Conversia unui număr subunitar din baza 10 în baza 2	107
Operații aritmetice cu numere scrise în baza 2	107
Adunarea a două numere binare	108
Scăderea a două numere binare	108
5. Memoria ca o tablă de scris	109
Memoria internă. Variabile de memorie	109
Organizarea memoriei interne	109
Variabila de memorie	110
Harta memoriei	110
6. Tipuri predefinite de date	111
Reprezentarea în virgulă mobilă a numerelor reale	112
7. Structura unui program Pascal	113
Declararea și inițializarea variabilelor	113
Inițializări implicite	114
Declararea constantelor	114
Expresii	114
Evaluarea unei expresii	114
Tipul unei expresii	115
8. Instrucțiuni de citire/scriere a datelor utilizând dispozitivele standard ale calculatorului: tastatura și monitorul	115
Instrucțiuni de citire a datelor — READ și READLN	115
Sintaxa instrucțiunilor de citire	116
Citirea unei singure valori de la tastatură	117
Citirea mai multor valori de la tastatură cu o singură instrucțiune de citire	118
Instrucțiuni de afișare a datelor — WRITE și WRITELN	119
Situatii de depășire	119
9. Instrucțiuni de citire/scriere a datelor utilizând fișiere text	120
Organizarea logică a unui fișier text	120
Operații de bază asupra fișierelor text	120
Identificarea unui fișier text	120
Deschiderea unui fișier text	120
Operații care se pot efectua asupra caracterelor dintr-un fișier text	121
Funcții predefinite	122
10. Bifurcații și intersecții în traficul unui program.	122
Instrucțiunea IF — o bifurcație în traficul programului	122

Sintaxa instrucțiunii IF	122
Mecanismul de funcționare a instrucțiunii IF	122
Tehnici de depanare. Execuția pas cu pas a unui program	127
Instrucțiunea CASE — o intersecție în traficul programului	137
Sintaxa instrucțiunii CASE	137
Mecanismul de funcționare a instrucțiunii CASE	138
IF versus CASE	138
11. Caruselul instrucțiunilor	141
O paralelă între instrucțiunile repetitive FOR, REPEAT și WHILE	141
Bucle de instrucțiuni	145
Sintaxa instrucțiunilor repetitive	145
Ieșirea forțată dintr-o buclă	146
Instrucțiunea BREAK	146
Instrucțiunea CONTINUE	146
Instrucțiunile EXIT și HALT	146
12. Aplicații cu instrucțiuni repetitive	146
Mulțimea divizorilor proprii ai unui număr natural	146
Strămoșul algoritmilor — algoritmul lui Euclid	149
Numărul de divizori ai unui număr natural dat	150
Cifrele unui număr natural	152
Codificare	153
Termenul general al unei sume	154
13. Numere magice	155
Numerele lui Nicomachos	155
Șirul Fibonacci	156
Numărul de aur	158
Puzzles cu numere Fibonacci	158
14. Produsul primelor n numere naturale	159
15. Arhitectură cu ... bytes	163
Tablouri unidimensionale (vectori)	163
Verificarea unei proprietăți a elementelor vectorului	165
Cea mai lungă secvență crescătoare	166
Capcanele lui ZERO	167
Codificarea informației	169
16. Numere uriașe	172
Povestea șahului	172
Produsul „uriaș” al două numere de tip longint	174
Secvența de calcul a diferenței dintre două numere „uriașe” $a = a-b$; $a > b$	175
Tablouri bidimensionale (matrice)	177
Legenda pătratului magic	177
Declararea unei matrice	177
Citirea matricei pe linii	177
Citirea matricei pe coloane	181
18. Șiruri de caractere (string)	181
Declararea și inițializarea unei variabile de tip string	182
19. Tipul înregistrare (record)	183
20. Arhitecturi complexe de date	184
În loc de recapitulare	185
Anexă	189
Bibliografie	

AUXILIARE ȘCOLARE

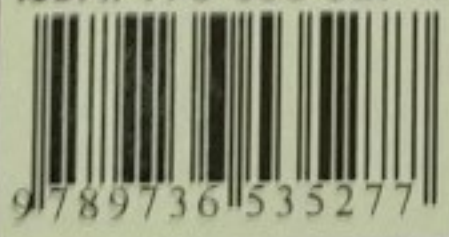
ELEMENTE DE PROGRAMARE lecții practice pentru elevii de gimnaziu

Prin aceste **lecții practice**, copiii sunt orientați, pas cu pas, să gândească rezolvarea problemelor cu calculatorul, de la analiza datelor și până la compunerea testelor care probează calitatea unui program. Lucrarea cuprinde atât probleme clasice, cât și probleme sau situații noi, adaptate vârstei. În alegerea și prezentarea acestora s-a urmărit și calitatea „produsului” realizat de programator: designul, ușurința în utilizare, performanța sau eficiența în timp a programului.

Conținutul lucrării este **accesibil** elevilor de gimnaziu, atât prin nivelul cunoștințelor prezentate, cât și prin stilul abordat, bazat pe comunicare directă, făcând apel la intuiție și creativitate.

Fiecare temă are un număr de idei, sfaturi practice sau trucuri utile oricărui programator. Pentru verificarea abilităților de gândire sau de programare dobândite, lucrarea conține algoritmi propuși spre descifrare sau programe care așteaptă să fie depanate (toate scrise în limbajul Pascal — cel mai folosit la nivel gimnazial).

ISBN: 973-653-527-4



9 789736 535277

120.000